UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE - UFRN
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

SOA-DB: Uma Arquitetura Embarcada Orientada a Serviço Para Acesso A Dispositivos Biomédicos

João Marcos Teixeira Lacerda

Natal, RN
junho de 2011

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
E DE COMPUTAÇÃO

SOA-DB: Uma Arquitetura Embarcada Orientada a Serviço Para Acesso A Dispositivos Biomédicos

João Marcos Teixeira Lacerda

Orientador: Prof. PhD. Ana Maria Guimarães Guerreiro

Co-orientador: Prof. D.Sc. Ricardo Alexsandro de Medeiros Valentim

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Natal, RN, junho de 2011

Catalogação na fonte. UFRN/Biblioteca Central Zila Mamede

Divisão de Serviços Técnicos

Lacerda, João Marcos Teixeira

SOA-DB: uma arquitetura embarcada orientada a serviço para acesso a dispositivos biomédicos / João MarcosTeixeira Lacerda.- Natal, RN, 2011.

71 f. : 40il.

Orientador: Ana Maria Guimarães Guerreiro
Co-Orientador: Ricardo Alexsandro de Medeiros Valentim

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-grraduação em Engenharia Elétrica.

1. Arquitetura orientada a serviços (Computador) – Dissertação. 2. Dispositivos biomédicos - Dissertação. 3. Sistemas embarcados (Computadores) – Dissertação. 4. Protocolo de comunicação - Dissertação. 5. Monitoramento – Pacientes – Dissertação. 6. Engenharia de Computação – Dissertação. I. Guerreiro, Ana Maria Guimarães. II. Valentim, Ricardo Alexsandro de Medeiros. III.Título.

UF/RN/BCZM CDU 004.2(043.3)

SOA-DB: Uma Arquitetura Embarcada Orientada a Serviço Para Acesso A Dispositivos Biomédicos

João Marcos Teixeira Lacerda

Dissertação de Mestrado aprovada em 30 de junho de 2011 pela banca examinadora composta pelos seguintes membros:

Profª. Drª. Ana Maria Guimarães Guerreiro (orientador) DEB/UFRN

Prof. Dr. Ricardo Alexsandro de Medeiros Valentim (co-orientador) DEB/UFRN

Prof. Dr. Gláucio Bezerra Brandão (Membro interno) DEB/UFRN

Prof. Dr. Luiz Felipe de Queiroz Silveira (Membro interno) DCA/UFRN

Prof. Dr. Filipe De Oliveira Quintaes (Membro externo) IFRN

Dedico este trabalho à minha família, pois sem esse suporte não conseguiria chegar aonde cheguei.

Agradecimentos

Agradeço aos professores Ricardo, Ana e Gláucio do DEB, a Aquiles e Luís Eduardo do PPGEEC e Samuel do DCA, por terém me dado um amplo suporte para o desenvolvimento deste trabalho.

Agradeço aos meus companheiros de laboratório Bruno, Jailton, Heitor, Vítor, Diegão, Jessé, Hélio, Desnes e João Paulo que não mediram esforços quando eu precisei de ajuda. Agradeço ao meu irmão Pedro, que sempre me ajudou nos conhecimentos avançados de inglês.

Agradeço à Tânia e Karla, respectivas mãe e namorada, por terem tido paciência nos momentos mais críticos.

Resumo

A grande diversidade na arquitetura de dispositivos biomédicos, aliada aos seus diferentes protocolos de comunicação, tem dificultado a implementação de sistemas que necessitam realizar o acesso a esses dispositivos. Diante dessas diferenças, surge a necessidade de prover o acesso a esses de forma transparente. Neste sentido, o presente trabalho propõe uma arquitetura embarcada, orientada a serviço, para acesso a dispositivos biomédicos, como forma de abstrair o mecanismo de escrita e leitura de dados nesses dispositivos, contribuindo desta maneira, para o aumento na qualidade e produtividade dos sistemas biomédicos, de forma a possibilitar com que, o foco da equipe de desenvolvimento de softwares biomédicos, seja quase que exclusivamente voltado aos seus requisitos funcionais.

Palavras-chaves: Monitoramento Remoto de pacientes, Dispositivos Biomédicos, Arquitetura Orientada a Serviço, Protocolos de Comunicação entre Dispositivos, Sistemas Embarcados.

Abstract

The great diversity in the architecture of biomedical devices, coupled with their different communication protocols, has hindered the implementation of systems that need to make access to these devices. Given these differences, the need arises to provide access to such a transparent manner. In this sense, this paper proposes an embedded architecture, service-oriented, for access to biomedical devices, as a way to abstract the mechanism for writing and reading data on these devices, thereby contributing to the increase in quality and productivity of biomedical systems so as to enable that, the focus of the development team of biomedical software, is almost exclusively directed to its functional requirements.

Keywords: Remote Monitoring of Patients, Biomedical Signals, Biomedical Devices, Service Oriented Architecture, Communication Protocols Devices, Embedded Systems.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

2D	Duas dimensões
3D	Três dimensões
A/D	Analógico-Digital
ARM	*Advanced RISC Machine*
ADSL	*Asymmetric Digital Subscriber Line*
API	*Application Programming Interface*
bps	*Bits por segundo*
CAD	*Computer-aided design*
CD-R	*Compact Disk Recordable*
CI	Circuito Integrado
DMIPS	*Dhrystone* MIPS
DSP	*Digital Signal Processor*
FAT	*File Allocation Table*
GUI	*Graphic Human Interface*
IEEE	*Institute of Electrical and Electronics Engineers*
I/O	*Input and Output*
IDE	*Interface Development Environment*
IRDA	*Infrared Data Association*
ISO	*International Organization for Standardization*
HTML	*Hiper Text Markup Language*
JRE	*Java Runtime Environment*
LPDDR	Low Power Double Data Rate
MIPS	Milhão de instruções por segundo
ms	milissegundos
OMAP	*Open Multimedia Application Platform*
PACS	*Picture Archiving and Communication Systems*
RFID	*Radio Frequency Identificator*
PC	*Personal Computer*
SIMD	*Single Instruction, Multiple Data*
SO	Sistema Operacional

SOA	*Service Oriented Architecture*
SoC	*System-on-a-chip*
SRAM	*Static Random Access Memory*
VFP	*Vector Floating Point*
RAM	*Random Access Memory*
TI	Tecnologia da Informação
TI	*Texas Instrument*
UART	*Universal Asynchronous Receiver/Transmitter*

Lista de publicações

Trabalhos completos publicados em anais de congressos:

Congresso	Título	Autores
	Mult-I/O - a middleware multi input and output for access devices: A case study applied the biomedical devices	João M. T. Lacerda; Bruno G. Araújo; Cicília R. M. Leite; Anna G. C. D. Ribeiro; Heliana B. Soares; Gláucio B. Brandão; Ricardo A. M. Valentim; Ana M. G. Guerreiro.
	Mult-I/O - Um Middleware Mult Entrada e Saída para Acesso a Dispositivos: Um Estudo de Caso Aplicado a Dispositivos Biomédicos	João M. T. Lacerda; Bruno G. Araújo; Cicília R. M. Leite; Jailton C. Paiva; Heliana B. Soares; Gláucio B. Brandão; Ricardo A. M. Valentim; Ana M. G. Guerreiro.
	SOA-DB: Uma Arquitetura Orientada A Serviço Para Acesso A Dispositivos Biomédicos	João M. T. Lacerda; Bruno G. Araújo; Gláucio B. Brandão; Ricardo A. M. Valentim; Ana M. G. Guerreiro.

Trabalho completo publicado em revista:

Revista	Título	Autores
R-BITS - Revista Brasileira de Inovação Tecnológica em Saúde	Um Middleware Como Interface Padrão Para Acesso A Dispositivos Biomédicos	João Marcos Teixeira Lacerda, Ricardo Alexsandro de Medeiros Valentim, Bruno Gomes Araújo, Gláucio Bezerra Brandão, Ana Maria Guimarães Guerreiro, Francis Solange Vieira Tourinho, José Diniz Júnior.

Capítulo 1

Introdução

A rápida disseminação de novos dispositivos de hardware no mercado tem aumentado a complexidade na comunicação entre esses. Segundo Valentim (2008), muitos dispositivos de hardware disponibilizam suas funcionalidades através de protocolos proprietários, de *drivers* que são dependentes de uma plataforma de sistema operacional. Diante desse ambiente fechado e heterogêneo de hardware, uma questão precisa ser discutida, a interoperabilidade. Isso porque, quanto mais divergentes forem, mais difícil e complexo serão os mecanismos de integração. Essa problemática está ainda mais presente na área biomédica, na qual muitos de seus dispositivos são proprietários e, portanto, dificultam o acesso e conseqüentemente o desenvolvimento de aplicações, como por exemplo, o monitoramento de pacientes, a aquisição de sinais biomédicos, entre outras.

Deugd et. al. (2006), relatou que o monitoramento e controle físico do ambiente tem sido possível através da disponibilização de interfaces comuns a dispositivos, desde básicos sensores e atuadores, a equipamentos digitais complexos.

Tais dispositivos, porém, são tradicionalmente dominados pelos desenvolvedores de sistemas embarcados. Diante desse fato, surge a necessidade de criar interfaces entre o mundo físico dos sensores e atuadores e o cenário de software dos sistemas corporativos.

Essa aproximação dos sensores e atuadores, dos sistemas corporativos, pode ser ainda mais justificada, pelo avanço tecnológico de dispositivos atuais, como placas baseadas em micro-controladores, com uma simples e eficaz interface de programação, a um baixo custo financeiro, Arduino (2011); E sistemas num único chip (*systems-on-a-chip*, Beagleboard (2011), que

conseguem reunir um microprocessador, memória RAM, interfaces de entrada e saída relativamente complexas, baixo consumo energético, resultando num dispositivo com poder de processamento bastante similar a um *netbook* ou *smartphone* de última geração.

Tal avanço desses dispositivos pode fornecer maiores possibilidades no âmbito da informática em saúde, por exemplo, aumentar a precisão de um oxímetro de pulso, melhorar a qualidade na visualização de um eletro encefalograma ou tornar possível a realização de um eletrocardiograma em tempo real, através da *WEB*.

Uma questão que pode ser levantada, considerando a melhoria nos serviços fornecidos por dispositivos médicos citadas anteriormente, é o aumento da complexidade no software embarcado nesses dispositivos. Esse aumento pode ser justificado pelo avanço tecnológico citado anteriormente nos dispositivos, na medida em que cada vez mais, softwares complexos, possam executar em dispositivos de menor tamanho físico, de menor consumo energético, a um menor custo financeiro.

Uma grande melhoria que esses novos dispositivos podem oferecer é o suporte a uma Arquitetura Orientada a Serviço (*Software Oriented Architecture – SOA*. SOA proporciona uma computação independente de protocolo, com baixo nível de acoplamento, Papazoglou (2007). A tecnologia dos *Web Services* é a implementação mais comum de *SOA* e considerada, por Schall et. al. (2005), como a evolução da *Web*, pois permite, mais do que a interação de seres humanos com aplicações via formulário HTML, a integração direta entre aplicações. Essa integração pode ser contextualizada no âmbito dos dispositivos médicos, visto que possibilita a integração tanto de aplicações médicas, como dispositivos médicos, pela razão desses implementarem a tecnologia dos *Web Services*, porém com versões adaptadas para a área médica de dispositivos embarcados, como proposto por Strähle et. al. (2007).

Além dessas qualidades, outra a ser buscada num dispositivo embarcado atual, é a capacidade de este fornecer recursos de rede, como as interfaces IEEE 802.03 (rede cabeada) e IEEE 802.11 (rede sem-fio), como uma maneira de acessar o dispositivo remotamente.

1.1. Objetivo

Um desenvolvedor de sistemas, ao implementar uma aplicação que precisa acessar um determinado dispositivo de hardware, se depara com alguns problemas:

- Mecanismos de acesso do dispositivo;
- Implementação dos métodos de acesso ao dispositivo;
- Lógica de negócio da aplicação passa a ficar em segundo plano.

Esses problemas serviram de motivação para o desenvolvimento da proposta SOA-DB, que efetivamente é uma camada de software que faz a intermediação entre as requisições e respostas direcionadas aos dispositivos biomédicos.

Diante desses fatos, este trabalho tem como objeto de estudo a proposta de uma arquitetura embarcada orientada a serviço, dinâmica e multi-paramétrica, devido à possibilidade de se conectar a diferentes dispositivos biomédicos, com diferentes interfaces de comunicação, aqui intitulada de SOA-DB. Essa arquitetura tem como proposta abstrair (tornar transparente) os mecanismos de escrita e leitura das aplicações clientes que acessam os dispositivos biomédicos, buscando, com essa abstração, alguns benefícios, como o aumento da produtividade de desenvolvimento, interoperabilidade e portabilidade.

O objetivo geral deste projeto é especificar, implementar, e validar uma arquitetura para abstrair o acesso a dispositivos biomédicos. Para tanto, utilizando como estudo de caso um sistema de monitoramento remoto de pacientes.

1.2. Objetivos Específicos

Os objetivos específicos deste trabalho de pesquisa consistem em:

- Realizar estudos sobre o monitoramento remoto de pacientes

- Propor uma arquitetura para abstrair o protocolo de acesso aos dispositivos biomédicos
- Diminuir a distância entre os cenários dos dispositivos médicos e os sistemas corporativos baseados nos conceitos de Tecnologia da Informação (TI).
- Desenvolver uma pesquisa entorno do objeto de estudo dessa dissertação de mestrado, de modo a elencar quais as tecnologias são as mais adequadas para a implementação da arquitetura proposta;
- Elicitar os requisitos da arquitetura a ser desenvolvida;
- Modelar a arquitetura;
- Implementar a arquitetura;
- Validar a arquitetura proposta através de uma análise de desempenho dirigida a testes de carga;

1.3. Estrutura do trabalho

Este trabalho encontra-se divido da seguinte forma:

O capítulo 2 descreve os principais temas relacionados ao trabalho desenvolvido. Inicia com uma abordagem sobre o Estado da arte deste trabalho, logo após trata de Arquitetura Orientada a Serviço e *Web Services*, *Axis*, Padrões de projeto orientado a objetos e Sistemas Embarcados, contendo as arquiteturas embarcadas mais utilizadas no contexto atual.

O Capítulo 3 especifica a SOA-DB e descreve todos os seus componentes, ilustra e descreve os ambientes de testes e aborda os resultados obtidos.

O Capítulo 4 trata das considerações finais do trabalho, bem como trabalhos futuros.

Capítulo 2

Pesquisa Bibliográfica

Neste capítulo, foram abordados os temas estudados no trabalho, iniciando com trabalhos recentes e relevantes para este trabalho, que serviram de base para tal. Logo após é tratado o Referencial Teórico, como SOA, *Web Services*, *Axis*, Padrões de Projeto, Sistemas Embarcados e por fim, Java Embedded SE.

2.1. Estado da Arte

No contexto de protocolos de acesso a dispositivos, Deugd (2006) propõe SODA – *Service Oriented Device Architecture*, como mecanismo para construção de interfaces abertas na forma de acesso (I/O – *Input and output*) aos dispositivos de qualquer natureza (Figura 1).

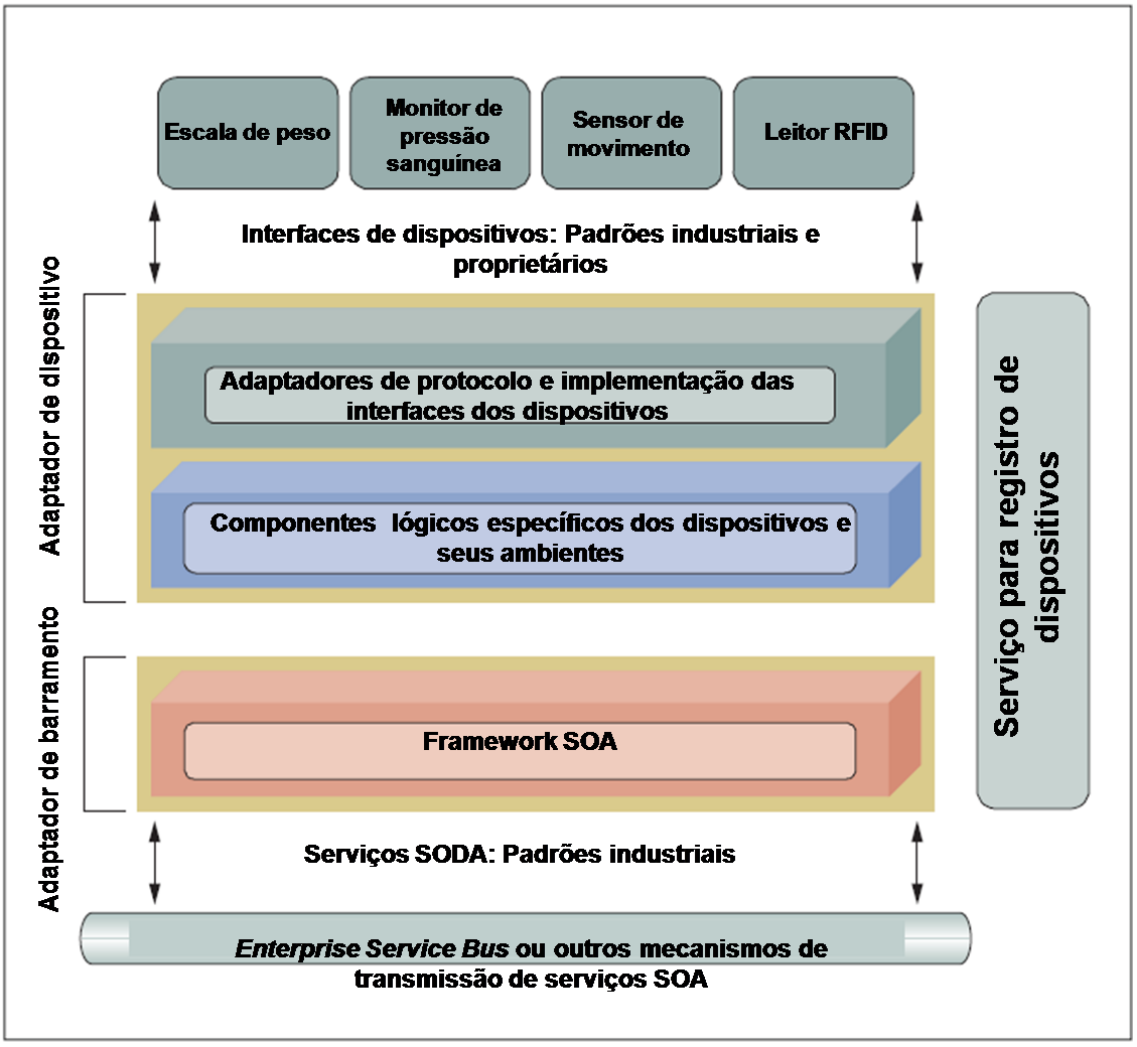

Figura 1: Descrição da SODA. Fonte: Adaptada de Deugd (2006).

Todavia, apesar de ser uma boa proposta, os autores de SODA ainda não a validaram como uma tecnologia consolidada. Nesta perspectiva, a proposta SOA-DB contribui também para demonstrar a viabilidade SODA através de testes experimentais.

No quesito monitoramento remoto de sinais biomédicos, Wang (2007) propõe um sistema de serviços inteligentes de enfermagem auxiliados por computador, de forma que equipamentos biomédicos são conectados num sistema embarcado, com a finalidade de coletar os sinais biomédicos de pacientes domésticos. Esse sistema, desenvolvido e implantado, tem cooperação de centros de monitoramento remoto. A Figura 2 ilustra o cenário de Wang (2007).

No cenário da Figura 2, pacientes domésticos utilizam uma *home health-care box* para o fornecimento e recepção da medição de sinais biomédicos, para o web *service* médico localizado no *care center*. Esse, por sua vez, compartilha essas medições com os hospitais, com o intuito de fornecer, o mais detalhadamente possível, essas informações ao clínico médico, dessa forma, aumentando a qualidade no seu diagnóstico.

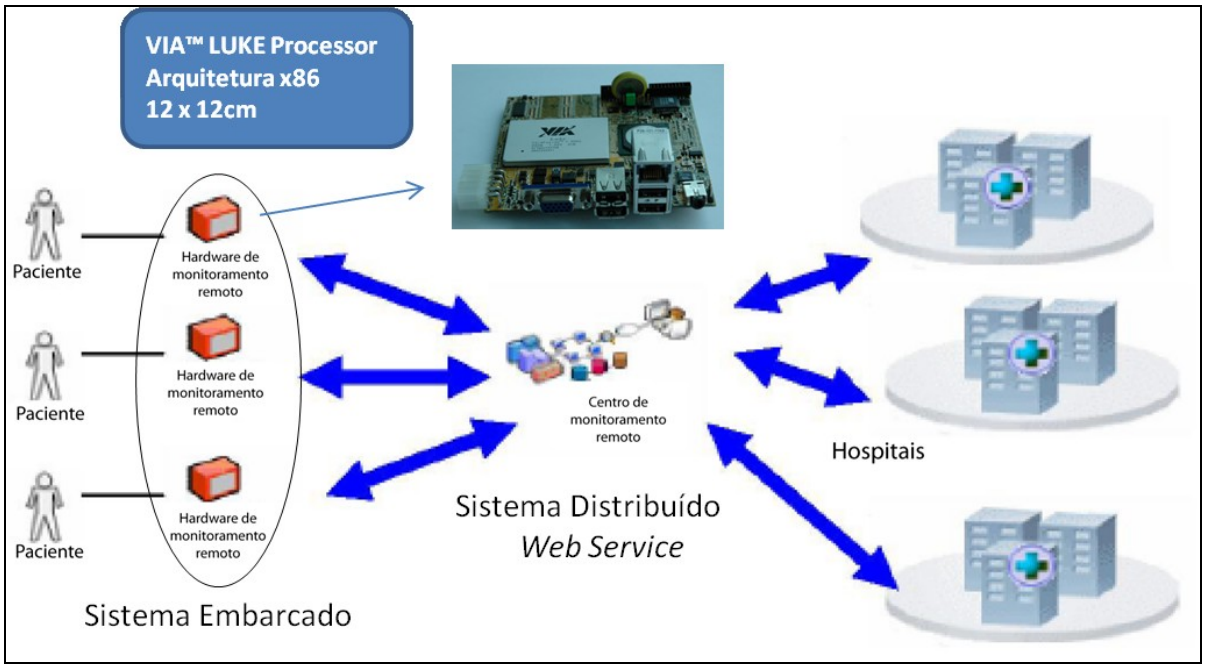

Figura 2: Cenário do sistema embarcado de monitoramento remoto de pacientes. Fonte: Adaptado de Wang et. al (2007).

Outro trabalho relevante no monitoramento de pacientes, de forma portátil, é proposto por De Capua (2010), como um sistema inteligente de medição de eletrocardiograma, baseado numa arquitetura de *Web Service*, para aplicações de tele medicina. Esse sistema, ilustrado na Figura 3, monitora e armazena os dados do paciente e, se ocorrer uma patologia, o sistema notifica um serviço de emergência.

No quesito SOA aliada a sistemas embarcados, Wolff (2007) propõe um *middleware* de rede, para arquiteturas orientadas a serviço, ao longo de sistemas embarcados heterogêneos. Esse trabalho aborda tecnologias existentes para realizar o *parsing* de XML e propõe uma nova tecnologia, intitulada de *uSOA*. Essa tecnologia tem o intuito de reduzir o tamanho e

processamento das mensagens XML. A Figura 4, Wolff (2007), ilustra esse cenário, com as tecnologias para a compressão de mensagens XML, visando à utilização em sistemas embarcados.

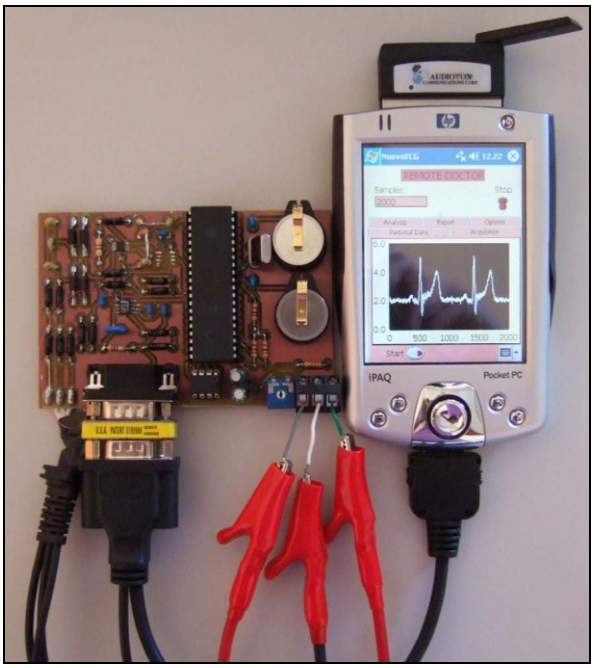

Figura 3: Dispositivo portátil conectado a uma placa de aquisição de sinais, constituindo um eletrocardiógrafo. Fonte: De Capua (2010).

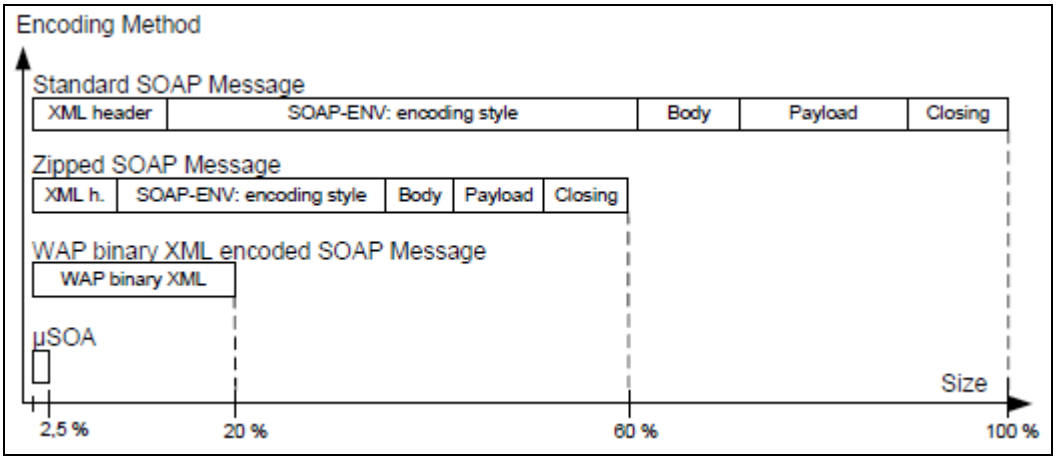

Figura 4: Arquitetura da SOA-DB. Fonte: Wolff (2007)

A partir da Figura 4, é possível perceber as reduções no tamanho dos pacotes de mensagens SOAP, que no caso da tecnologia *Zip* foi de 40%, na

tecnologia de codificação *WAP Binary XML* foi de 80% e por fim, na tecnologia *uSOA*, houve uma redução significante de 97,5% no pacote SOAP.

2.2. SOA

Segundo Papazoglou (2007), SOA (*Service Oriented Architecture* – Arquitetura Orientada a Serviço) é uma forma emergente que aborda requisitos de baixo acoplamento, baseada em padrões e uma computação distribuída independente de protocolo. Tipicamente, operações de negócio rodando em SOA, abrangem um número de invocações desses diferentes componentes, freqüentemente em uma orientação a eventos ou de forma assíncrona, que refletem as necessidades de um processo de negócio.

Para a implementação de SOA é requerido uma alta comunicação e integração de *backbones* distribuídos. Esta funcionalidade é provida pelo *Enterprise Service Bus* (ESB) que é uma plataforma de integração que utiliza padrões *Web Services* para suportar uma grande variedade de padrões de comunicações ao longo de múltiplos protocolos de transporte e fornecer recursos de valor agregado para aplicações SOA. Papazoglou (2007) revisa tecnologias e abordagens que unifica os princípios e conceitos de SOA com aqueles da programação baseada em eventos.

Além do que SOA possui padrões que estendem a funcionalidade de um *middleware*, conectando sistemas e componentes heterogêneos e que oferece serviços de integração.

2.2.1. Arquitetura básica SOA

Papazoglou (2003) relata que uma SOA básica não é uma arquitetura que trata apenas de serviços, ela é um relacionamento de três tipos de participantes: O provedor de serviço, a agência de descoberta de serviços e o requisitante de serviço (cliente). As interações envolvem a **publicação**, **procura** e **vinculação** de operações, ver na Figura 5. Essas regras e operações agem em torno dos artefatos dos serviços (A descrição e implementação do serviço).

Em um típico cenário baseado em serviço, um provedor do serviço hospeda um módulo de software de rede acessível (uma implementação de um dado serviço). O provedor do serviço define uma descrição de serviço do serviço e a publica em um cliente ou agência de descoberta de serviço, de modo que uma descrição do serviço se torna detectável. O cliente do serviço utiliza uma operação de busca para recuperar a descrição do serviço de uma agência de serviço conhecida, tal como um registro ou repositório, que utiliza essa descrição para vinculá-la com o provedor de serviço e invocá-lo com o propósito de interagir com sua implementação. As regras do provedor e requisitante de serviço são construções lógicas, o que acarreta um serviço poder assumir características de ambos.

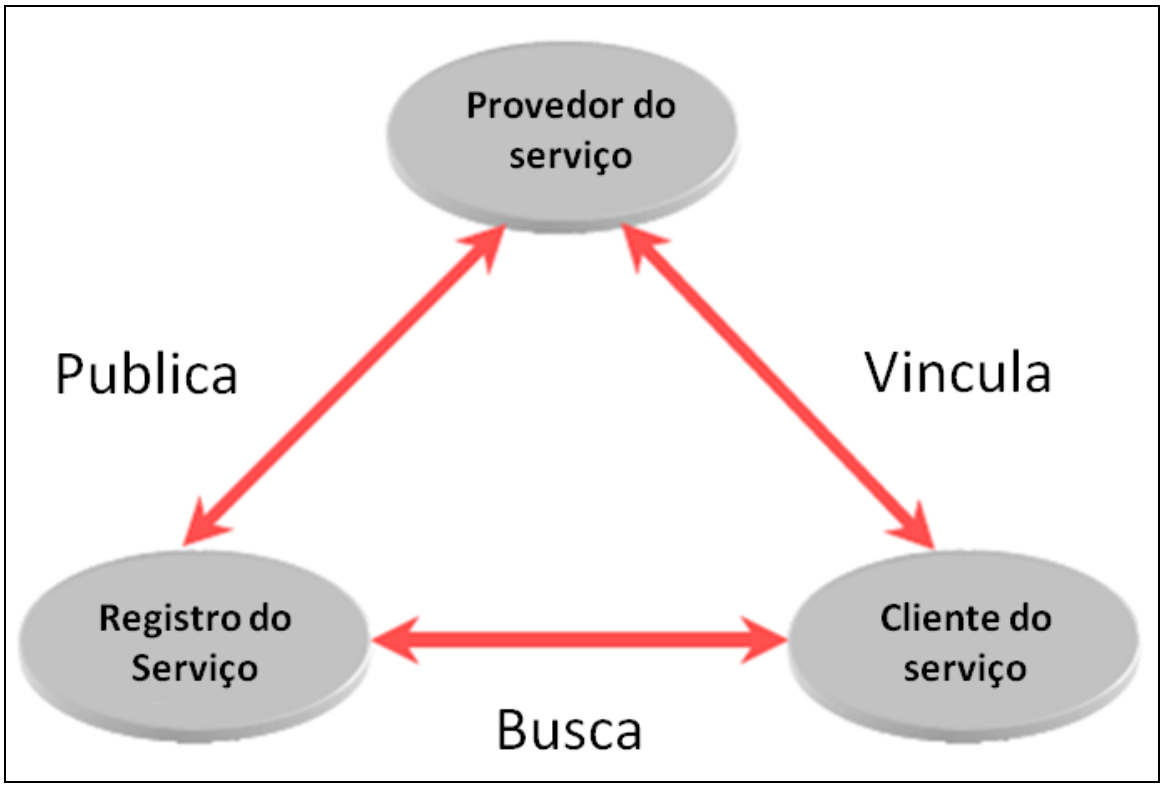

Figura 5: Relacionamento entre os conceitos SOA. Fonte: Papazoglou (2003).

2.3. *Web Services*

Um conceito bem definido para *web services* é descrito pela W3C (2008): “Um *web service* é uma aplicação de software identificada por uma URL, cujas interfaces e ligações são capazes de serem definidas, descritas e

descobertas por artefatos XML. Um *Web service* suporta integrações diretas com outros agentes de software, usando mensagens baseadas em XML, trocadas via protocolos baseados em internet."

De acordo com Papazoglou (2007), os *web services* se tornaram a implementação preferida para cumprir as promessas de SOA, às quais podem ser citadas: O máximo compartilhamento de serviço, o reuso e a interoperabilidade. *Web Services* reduzem a complexidade de aplicações corporativas por encapsulamento e minimizam os requisitos para a compreensão compartilhada, por definir interfaces de serviço de uma maneira clara e precisa. *Web services* também possibilitam integração em tempo real e entre aplicações legadas. Por ser baseado em padrões abertos e difundidos, Papazoglou (2007) conclui que os *Web Services* parecem ser preparados para o sucesso, uma vez que são construídos baseados no que já existe, como em HTTP, SOAP, W3C (2001) e XML, Vidgen et. al. (2000).

2.3.1. Tecnologias envolvidas

Além do XML, os *web services* são compostos pelas seguintes tecnologias:

a) SOAP (*Simple Object Access Protocol*): É um protocolo baseado em XML para troca de mensagens. Provê um leve e simples mecanismo para a troca de informações em um ambiente distribuído (W3C, 2000).

b) WSDL (*Web Services Description Language*): Essa linguagem provê um modelo para descrever os *web services*. Ela possibilita a separação das funcionalidades abstratas oferecidas por um serviço, das funcionalidades concretas, tais como "onde" e "como" elas são oferecidas (W3C, 2001).

c) UDDI (*Universal Description, Discovery, and Integration*): No contexto dos *web services*, o protocolo UDDI define um padrão para publicar e descobrir os serviços dentro de uma arquitetura orientada a serviço (UDDI, 2004).

Com as tecnologias citadas acima, pode-se descrever como elas se relacionam. Erl (2005) expõe este relacionamento através da Figura 6:

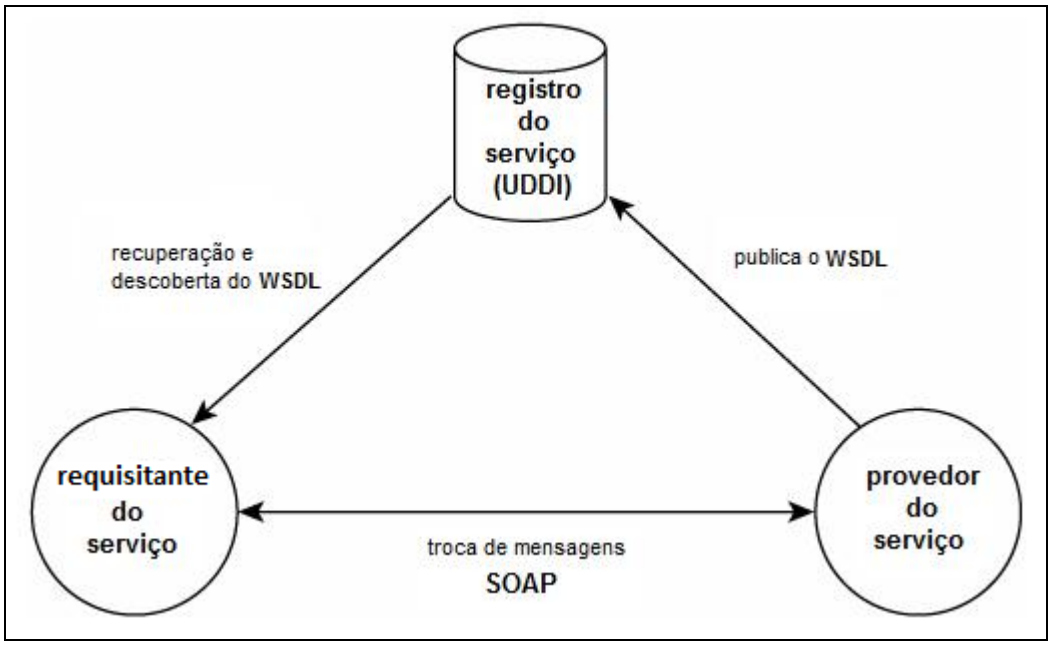

Figura 6: Relacionamento entre os conceitos de SOA e suas tecnologias.
Fonte: Adaptado de Erl (2005)

Endrei (2004) ainda cita algumas características inerentes aos *web services* tidas como características chave, possibilitando a esses serem uma alternativa apropriada para a implementação de uma arquitetura orientada a serviço. Dentre essas características podem ser citadas:

a) **Auto-suficiência**: Nenhum software adicional é necessário no lado cliente da aplicação, apenas uma linguagem de programação que suporte XML e HTTP do lado cliente. Pelo lado servidor, apenas um servidor web e uma engenharia de *servlet* são necessários. É possível que o *web service* utilize uma aplicação qualquer sem que seja necessária a produção de nenhuma linha de código.

b) **Auto-descrição**: Devido ao fraco acoplamento entre as aplicações, nem o cliente tampouco o servidor conhecem o formato ou conteúdo da mensagem. Definições sobre o conteúdo da mensagem trafegam junto com ela, além de não serem necessários repositórios de metadados externos ou ferramentas de geração de código.

2.3.2. Axis (*Apache Extensible Interaction System*)

De acordo com a Apache (2011), Axis é essencialmente um projeto voltado para o protocolo SOAP – um framework SOAP que constrói processos SOAP como clientes, servidores, gateways, etc. A versão atual do Axis está escrita em Java, mas uma implementação em C++ do lado cliente do Axis está sendo desenvolvida.

- O Axis não é só uma *engine* SOAP – ele também inclui:
- Um simples servidor *stand-alone;*
- Um servidor que se integra com uma *engine* de *servlets*, tal como o contêiner web Java Tomcat, Apache (2005-2);
- Suporte extensivo para WSDL;
- Ferramenta que gera classes Java a partir de um WSDL;
- Alguns programas exemplo;
- Uma ferramenta para monitorar pacotes TCP/IP.

Axis é a terceira geração do Apache SOAP (que começou com o "SOAP4J" da IBM). Em meados de 2000, os comitês da Apache SOAP v2 começaram a discutir como tornar esta *engine* mais flexível, configurável e apta a manipular tanto o SOAP quanto a próxima especificação do protocolo XML da W3C.

Depois de certo tempo, tornou-se claro que era necessário um upgrade da *engine*. Alguns dos comitês v2 propuseram projetos muito parecidos, todos baseados em "cadeias" de mensagens "manipuláveis" que poderiam implementar pequenas funcionalidades de bits de uma forma bastante flexível e combinável.

Depois de meses de discussões e esforços de programação, o Axis agora disponibiliza os seguintes recursos:

- **Velocidade**: O Axis utiliza o SAX, um conversor baseado em eventos que possui uma velocidade significativamente maior do que versões anteriores do Apache SOAP.
- **Flexibilidade**: A arquitetura do Axis dá ao desenvolvedor completa liberdade para inserir extensões da *engine* para o processamento de

cabeçalho personalizado, gerenciamento do sistema ou qualquer outra coisa que se queira.

- **Estabilidade**: Axis define um conjunto de interfaces publicadas que mudam sutilmente comparadas com o resto do Axis.
- **Implantação orientada a componentes**: Pode-se definir redes reusáveis de manipuladores para implementar padrões compartilhados de processamento de aplicações ou para a distribuição para parceiros.
- **Framework de transporte**: Tem-se uma abstração “limpa” e simples para se projetar o transporte (ex.: remetentes e ouvintes do SOAP ao longo de vários protocolos como SMTP, FTP, *middleware* orientado a mensagens, etc.), adicionalmente, o núcleo da engine é completamente independente de transporte.
- **Suporte a WSDL**: O Axis suporta o WSDL versão 1.1, que permite, com simplicidade, tanto a criação de *stubs* para o acesso a serviços remotos, quanto exportar, de forma automática, descrições legíveis por máquinas de seus serviços implantados.

2.4. Padrões de projeto Orientado a Objetos

Gamma (1995) cita a afirmação de Christopher Alexsander sobre padrões de projeto: “Cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira”. A frase de Alexsander foi aplicada a padrões voltados para a área de engenharia civil, mas que também pode ser utilizada no conceito de software orientado a objetos.

Dessa forma, Gamma et. al. (2000) define padrões de projeto orientado a objetos como: “Descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular”.

Nesse sentido, foram definidos quatro elementos essenciais para cada padrão:

1- **Nome do padrão**: Elemento para identificá-lo na literatura.

2- **O problema**: Explica em que situação aplicar o padrão, o problema e seu contexto.

3- **A solução**: Descreve os elementos principais do padrão, tanto como seus relacionamentos, responsabilidades e colaborações.

4- **As conseqüências**: São os resultados e análises das vantagens e desvantagens da aplicação do padrão.

2.4.1. Abstract Factory

Gamma (1995) define o padrão *Abstract Factory* como um padrão que provê uma interface para a criação de objetos relacionados ou dependentes, sem a necessidade de especificação de suas classes concretas.

Este padrão é contextualizado através de um kit de interface gráfica com o usuário, na qual seus produtos, como janelas, botões e barras de rolagem possuem várias opções de aparência e comportamento. Uma aplicação para ser portável, levando em consideração esses diversos comportamentos, ela precisa ter uma política de codificação pouco complexa. A instanciação de classes específicas aos utilitários da GUI pode acarretar uma difícil mudança de comportamento no futuro.

Esse problema pode ser resolvido pela definição de uma classe abstrata **FabricaUtilitario** (Figura 7) que declara uma interface para criar cada tipo básico de utilitário. Existe uma classe abstrata para cada tipo de utilitário e subclasses concretas implementam utilitários específicos para diferentes padrões de aparência e comportamento. A interface **FabricaUtilitario** tem um método que retorna um novo objeto “utilitário” para cada classe abstrata **Utilitario**. Clientes chamam esses métodos com a finalidade obter instâncias de utilitários, porém eles não têm o conhecimento das classes concretas que as utilizam. Desta forma, os clientes se tornam independentes dos detalhes de implementação dos utilitários.

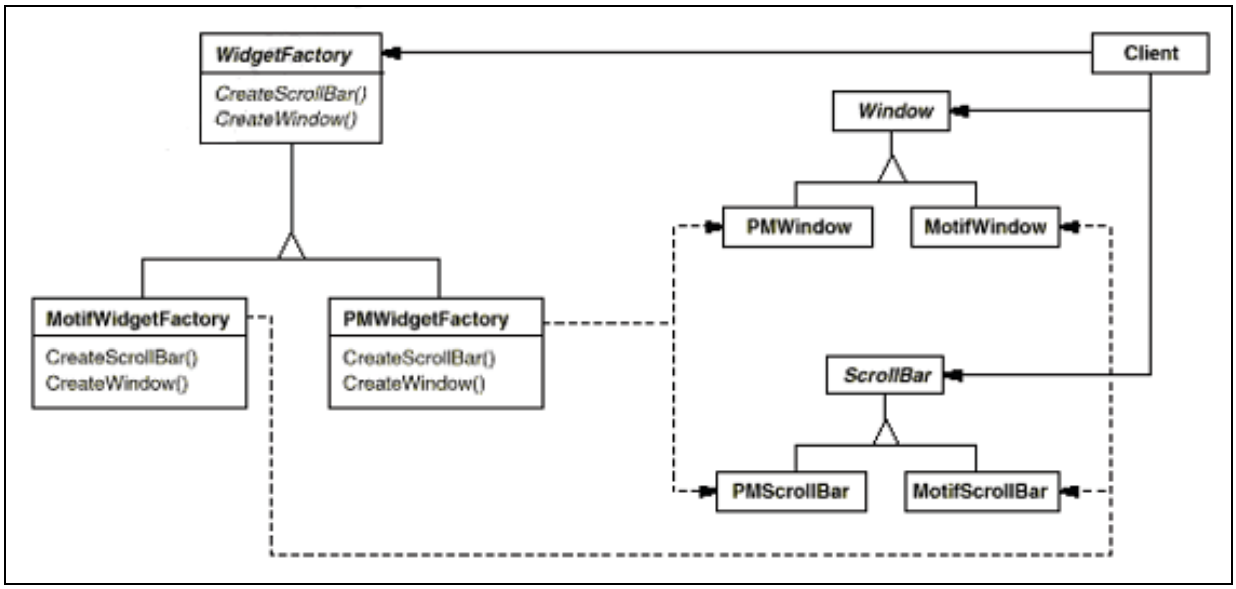

Figura 7: Cenário do padrão de projeto *Abstract Factory.* Fonte: Gamma (1995).

Existe uma subclasse concreta de **FabricaUtilitario** para cada padrão de aparência e comportamento. Cada subclasse implementa métodos para criar o utilitário apropriado para a aparência e comportamento desejados. Por exemplo, o método criarBarraRolagem na FabricaUtilitarioAssunto instancia e retorna uma barra de rolagem **Assunto**, enquanto a operação correspondente na **GAFabricaAssunto** retorna uma barra de rolagem para Gerenciador de Apresentação.

Clientes instanciam utilitários somente através da interface **FabricaUtilitario**, e não tem conhecimento das classes que implementam utilitários para um específico padrão de comportamento e aparência. Em outras palavras, clientes precisam apenas se comprometer com uma interface definida por uma classe abstrata, não com uma classe concreta em particular.

Uma **FabricaUtilitario** também força dependências entre as classes concretas de utilitários. Uma barra de rolagem **Assunto** precisa ser utilizada com um botão **Assunto** e um editor de texto **Assunto**, de maneira que essa restrição é aplicada automaticamente como conseqüência da utilização de uma **FabricaUtilitarioAssunto**.

Este padrão de projeto é utilizado quando:

- Um sistema precisa ser independente de como seus componentes são criados, compostos e representados.
- Um sistema precisa ser configurado com múltiplas famílias de produtos.
- Uma família de objetos de componentes relacionados é designada a serem utilizadas em conjunto e é preciso forçar essa dependência.
- Quando se quer prover uma biblioteca de classes de produtos, e se deseja revelar apenas suas interfaces, não suas implementações.

As conseqüências da utilização do padrão de projeto Abstract Factory são:

- Ele isola classes concretas: O padrão *Abstract Factory* ajuda o desenvolvedor a controlar classes de objetos que uma aplicação cria. Devido à fábrica encapsular a responsabilidade e o processo de criação de objetos componentes, ocorre o isolamento do cliente da implementação das classes. Os Clientes manipulam instâncias através de suas interfaces abstratas. Os nomes das classes de produtos são isolados na implementação da fábrica concreta; eles não aparecem no código cliente.
- Ele torna fácil a mudança de produtos de famílias. A classe de uma fábrica concreta aparece apenas uma vez na aplicação – isto é, quando ela é instanciada. Isto permite mudar de forma fácil uma aplicação que utiliza uma fábrica concreta. Ela pode utilizar diferentes configurações de produtos simplesmente mudando a fábrica concreta. Devido a uma fábrica abstrata criar uma família concreta de produtos, as alterações em uma família inteira de produtos ocorrem uma única vez. Em nossa GUI, por exemplo, nós podemos mudar de utilitários **Assunto** para **GerenciadorApresentação** simplesmente pela troca de objetos fábrica.
- Ele promove consistência entre produtos. Quando objetos de produtos em uma família são designados a trabalharem juntos, é importante que uma aplicação use objetos de apenas uma família ao mesmo tempo. *AbstractFactory* torna isso fácil de ser amarrado.

- Suportar novos tipos de produtos é difícil. Estender fábricas abstratas para produzirem novos tipos de produtos não é uma tarefa fácil. Isso ocorre porque a interface **AbstractFactory** fixa o conjunto de produtos que podem ser criados. Suportar novos tipos de produtos requer estender a interface Factory, que envolve modificar a classe **Abstractfactory** e todas as suas subclasses.

2.4.2. Factory Method

Gamma (1995) define este padrão de projeto como uma interface para a criação de um objeto, porém fica a cargo das subclasses decidirem qual classe instanciar. O Factory Method deixa uma classe diferir a instanciação de suas subclasses.

O padrão de projeto *Factory Method* é comparado a um *framework (achar outra referência para framework)*, que utiliza classes abstratas para definir e manter relacionamentos entre objetos.

Considerando um *framework* de aplicações (Figura 8) que pode apresentar múltiplos documentos para o usuário. Duas abstrações chave nesse framework são as classes **Aplicação** e **Documento**. Ambas as classes são abstratas e os clientes têm que especializá-las para realizar suas implementações específicas. Para criar uma aplicação de desenho, por exemplo, são definidas duas classes, **DesenhoAplicativo** e **DesenhoDocumento**. A classe **Aplicativo** é responsável por gerenciar os documentos e criar o que elas necessitam – quando o usuário seleciona Abrir ou Novo no *menu*, por exemplo.

Devido à subclasse específica **Document**o instanciar um aplicativo específico, a classe Aplicativo não pode predizer a subclasse de Documento a instanciar – a classe Aplicativo apenas sabe quando um novo documento deve ser criado, não que tipo de Documento criar. Isso cria um dilema: O framework deve instanciar classes, mas ele conhece apenas as classes abstratas, às quais não podem ser instanciadas.

O padrão de projeto *Factory Method* encapsula o conhecimento das subclasses **Documento** para manipular esse conhecimento fora do framework.

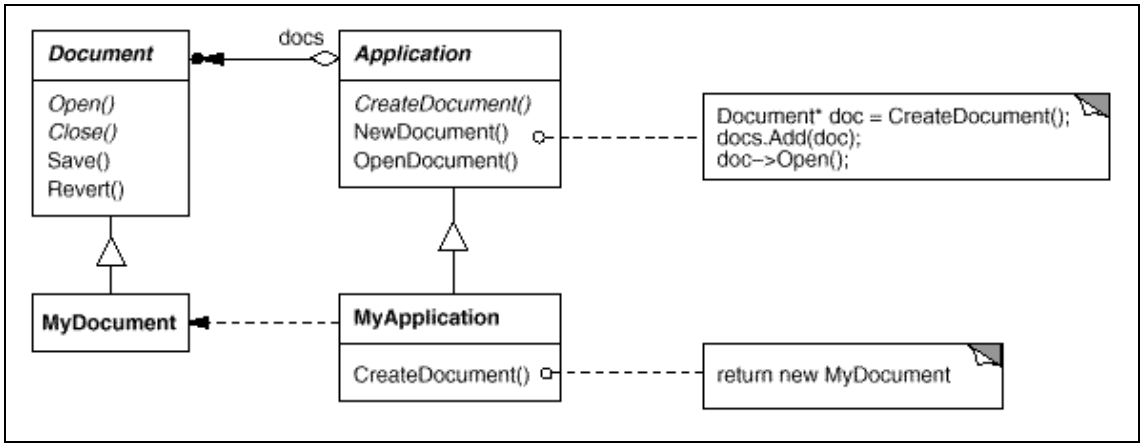

Figura 8: O padrão de projeto Factory Method encapsula o conhecimento das subclasses **Documento** para manipular esse conhecimento fora do framework. Fonte: Gamma (1995).

2.5. Sistemas embarcados

Segundo Zurawski (2004), o mundo dos eletrônicos tem testemunhado um crescimento dramático de aplicações embarcadas nas últimas décadas. Das telecomunicações ao entretenimento, de automóveis a bancos, em quase tudo no nosso dia-a-dia é possível verificar a presença de componentes eletrônicos.

Na maioria dos casos, esses componentes são baseados em sistemas de computação, que não são, entretanto, usados ou percebidos como computadores. Por exemplo, eles geralmente não possuem um teclado ou um display para interagirem com o usuário, assim como não executam aplicações e sistemas operacionais usuais.

Às vezes, esses sistemas constituem um sistema autônomo (ex.: um celular), mas eles são freqüentemente embarcados em outro sistema, por serem providos de melhores funcionalidades e desempenho (ex.: a unidade de controle do motor de um veículo). Esses sistemas de computação são chamados de **sistemas embarcados**.

O enorme sucesso dos eletrônicos embarcados se deve a algumas causas. A principal, segundo Zurawski (2004), é possibilitar um aumento exponencial no desempenho e funcionalidade, a um custo cada vez menor.

Esse fato é possível por duas razões: Primeiro, devido ao alto desempenho de um CI, graças a investimentos maciços em tecnologia e métodos de fabricação, que permitem uma forma de se fabricar dispositivos cada vez mais complexos. Segundo, pelo desenvolvimento de novas metodologias de projeto, que possibilitam eficiência e inteligência no uso desses dispositivos.

Existem muitos exemplos de sistemas embarcados no mundo real. Por exemplo, um carro moderno contém dezenas de componentes eletrônicos (unidades de controle, sensores e atuadores) que realizam tarefas variadas.

O primeiro dos sistemas embarcados que surgiu num carro está relacionado à sua transmissão e controle de suspensão, tais como o controle do motor, o sistema de freios antitravamento o controle de transmissão e suspensão propriamente dito.

Na área das telecomunicações, por exemplo, um telefone celular é um sistema embarcado do qual o seu ambiente é a rede móvel. Eles são computadores muito sofisticados no qual sua tarefa principal é enviar e receber voz, porém são freqüentemente utilizados como assistentes pessoais digitais, para jogos, imagens, mensagens multimídia e navegação na internet sem fio. Eles têm sido tão bem sucedidos e pervasivos que, em uma década, se tornaram essenciais na vida das pessoas.

Outros tipos de sistemas embarcados mudaram significativamente as vidas das pessoas, por exemplo, as maquinetas de cartão de crédito e débito, modificaram a forma como as pessoas realizam pagamentos. Os *players* digitais de mídia também mudaram o modo como as pessoas escutam músicas ou assistem a vídeos.

No ano de 2004, percebeu-se o início de uma revolução que causaria um forte impacto na maioria dos setores industriais. Através da proliferação dos sistemas embarcados, os quais seriam encontrados na maioria dos objetos de uso diário das pessoas.

Os sistemas embarcados são flexíveis quanto à mudança de ambiente. A maioria deles também possui conexão sem-fio, com o intuito de acompanhar as pessoas aonde quer que estas forem, e mantê-las constantemente

conectadas com a informação que precisam e com outras pessoas nas quais se relacionam.

A revolução citada nos parágrafos anteriores mudará até mesmo o papel dos computadores no cenário mundial, tal como muitas das aplicações que são utilizadas hoje serão desempenhadas por sistemas embarcados especializados. Além disso, as indústrias também precisarão dominar cada vez mais o projeto de sistemas embarcados, ou terão que terceirizar esse serviço, principalmente as que são voltadas para aplicações em tempo real.

Outra revolução similar vem acontecendo em outras áreas além das socioeconômica e industrial, como nas áreas de entretenimento, turismo, educação, agricultura, governos, entre outras. Portanto, é evidente que novas e mais eficientes metodologias de projeto de eletrônicos embarcados precisam ser desenvolvidas, de maneira a permitir a indústria fazer uso dessa tecnologia.

Sistemas embarcados são informalmente definidos, por Zurawski (2004), como uma coleção de partes programáveis cercados por “aplicações específicas de circuitos integrados” (*Application Specific Integrated Circuits – ASICs*) e outros componentes padrões, chamados de “porções padrões de aplicativos específicos” (*Application Specific Standard Parts - ASSPs*) que interagem continuamente com um ambiente através de sensores e atuadores.

Essa coleção pode ser fisicamente um conjunto de chips numa placa ou um conjunto de módulos em um CI. O software num sistema embarcado é utilizado para recursos e flexibilidade, enquanto seu hardware dedicado é utilizado para um aumento de desempenho e redução no consumo de energia. Um exemplo de arquitetura de um sistema embarcado é mostrado na Figura 9.

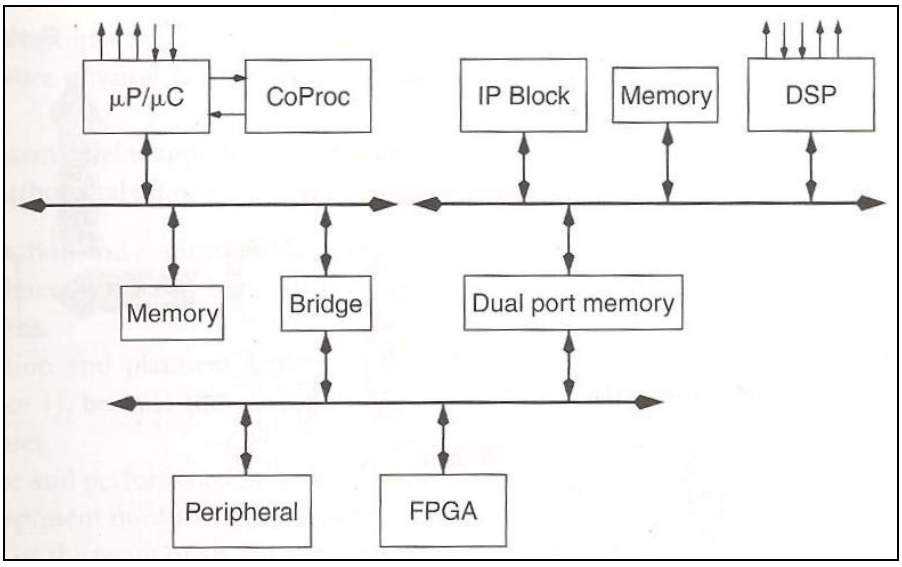

Figura 9: Arquitetura de um Sistema Embarcado. Fonte: Zurawski (2004).

No cenário ilustrado na Figura 9, os principais componentes programáveis são os microprocessadores e os processadores digitais de sinais (*Digital Signal Processors* – DSPs), que implementam a porção de software do sistema. Podem-se ver componentes reconfiguráveis, especialmente se eles podem ser reconfigurados em tempo de execução. Eles exibem características de área, custo, desempenho e de consumo, que são intermediados entre hardware dedicado e processadores.

Componentes de hardware programáveis e customizáveis, por outro lado, implementam blocos específicos de aplicação e periféricos. Todos os componentes são conectados através de redes, barramentos padrões e dedicados, e o dado é armazenado num conjunto de memórias. Geralmente alguns subsistemas menores são interligados para o controle, por exemplo, um celular constituído de uma rede sem-fio.

Além da idéia descrita na Figura 9, existem vários conceitos para Sistemas Embarcados. Por exemplo, Heath (1998) afirma que “Um sistema embarcado é designado para um propósito específico, não podendo ser programado pelo usuário da mesma forma que um computador pessoal e que o usuário pode fazer escolhas referentes à funcionalidade, mas não pode mudar esta funcionalidade pela adição ou substituição de software.

Esse conceito em 2003 talvez fosse aceitável, porém, atualmente existem diversos dispositivos considerados embarcados que confrontam essa afirmação, como os *smartphones*, que fornecem a possibilidade da adição de softwares, por meio de aplicações que podem ser descarregadas na *Internet* e instaladas a qualquer momento. Esse fato só reforça o quão flexível o conceito de Sistema Embarcado tem se tornado ao longo do tempo.

Um conceito bastante atualizado de Sistemas Embarcados pode ser encontrado no sítio da Wikipédia (2010): “Um sistema embarcado (ou sistema embutido) é um sistema micro processado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferente de computadores de propósito geral, como o computado pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos”.

Tomando como base esse conceito da Wikipédia, qualquer dispositivo que possua poder de processamento e que seja destinado a um fim específico, pode ser considerado um Sistema Embarcado.

2.6. Os chips ARM

Segundo Morimoto (2010), a migração dos chips PowerPC para os x86 feita pela Apple em 2005, tornou os chips x86 cada vez mais presentes nos computadores pessoais. A escolha do fabricante pode variar entre Intel, AMD ou VIA, porém o legado das instruções vindas desde o processador 8086 continuará presente.

Enquanto que os chips x86 são bastante visíveis, o seu uso em larga escala é bem menor comparado aos chips ARM (*Advanced Risk Machine*). Mesmo com uma maior visibilidade, os chips x86 ficam muito atrás em termos de uso dos chips ARM, através dos Z80, que são vendidos aos bilhões e utilizados em todo o tipo de dispositivos eletrônicos.

Atualmente, os chips ARM são utilizados em uma gama de dispositivos eletrônicos, como celulares, *smartphones*, roteadores ADSL e vídeos-game portáteis. De acordo com Morimoto (2010), basicamente os chips x86 são

utilizados em PCs, *notebooks* e *netbooks*, enquanto os ARM são utilizados em praticamente todo o resto.

O ponto chave para o grande número de funções designadas pelos chips ARM, baixo consumo e baixo custo é a integração dos componentes, acompanhadas pelo uso de controladores dedicados para diversas funções; diferente de um PC, onde quase tudo é feito pelo processador principal.

Os chips ARM são projetados por uma única empresa, a *ARM Holdings,* Furber (2000), porém licenciados e produzidos por diversos fabricantes. A *ARM Holdings* é responsável pelo desenvolvimento dos chips e detentora dos direitos sobre a arquitetura, porém não produz os processadores, se limita a licenciar os projetos a preços módicos para outros fabricantes, que podem escolher por diferentes tipos de licenças, incluindo funções que permitem a alteração dos chips e inclusão de novos componentes. Fabricantes como a QualComm, Texas Instruments, Apple e a Samsung desenvolvem suas soluções proprietárias dos chips ARM, incluindo controladores auxiliares e modificações diversas.

Segundo Morimoto (2010), os processadores ARM podem ser divididos em duas grandes famílias. A primeira é a dos chips “AMRX”, dos quais os principais da atualidade são os AMR7, ARM9 e ARM11 (que são mais antigos, porém ainda muito utilizados); e a família *Cortex*, composta pelo Cortex A5, Cortex A8 e Cortex A9, representando a geração mais atual.

2.6.1. A linha Cortex A

A linha ARM Cortex–A para processadores de aplicações provê uma grande variedade de soluções para dispositivos, controlando uma rica plataforma de SO e aplicações de usuário, que vão desde aparelhos de baixo custo como smatphones, plataformas de computação móvel, TV digital e set-top boxes e soluções de impressão e servidores para redes corporativas.

O alto desempenho do Cortex-A15, o escalável Cortex-A9, o comprovado pelo mercado Cortex-A8 compartilham a mesma arquitetura e, portanto, uma completa compatibilidade de aplicativos, que inclui suporte à

tradicional linha “ARMX”, o conjunto de instruções *Thumb*, e o novo conjunto de instruções *Thumb-2*, compacto e de alto desempenho.

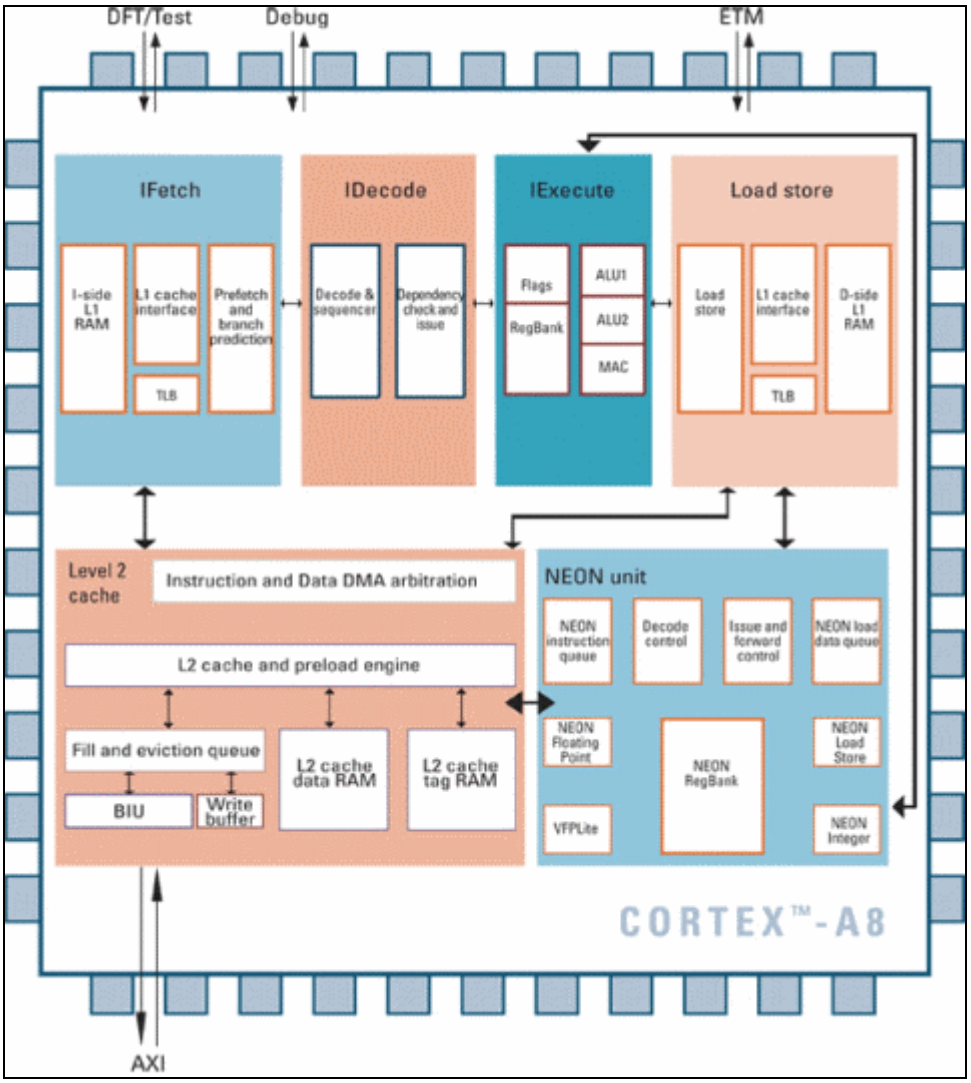

Figura 10: Arquitetura do Cortex-A8. Fonte: ARM (2011).

Cortex-A8: O ARM Cortex-A8 é baseado na arquitetura ARMv7, proposta no em ARM (2011) e suporta velocidades desde 600 MHz a mais de 1GHz. O processador Cortex-A8 tem a capacidade de balancear os requisitos de consumo otimizado dos dispositivos móveis, que operam a menos do que 300mW, tanto quanto otimizado para desempenho de aplicações consumidoras que necessitam 2000 Dhrystone MIPS.

O processador de alto desempenho Cortex-A8 está amplamente difundido entre os dispositivos atuais. Desde celulares de última geração a *netbooks*, DTVs, impressoras e linha automotiva, este processador oferece uma solução de alto desempenho comprovada, tendo milhões de unidades vendidas anualmente. A Figura 10 ilustra a arquitetura do Cortex-A8.

2.6.2. Sistema-em-um-chip

Segundo Júnior (2005), *System-on-a-chip* (SoC), *System On Chip* (SOC) ou, em português, sistema-em-um-chip, se refere a todos os componentes de um computador, ou qualquer outro sistema eletrônico, em um circuito integrado (chip). Ele pode conter funções digitais, analógicas, de sinais mistos e muitas vezes de freqüências de radio; Ou seja, várias funções em uma única placa de silício.

De acordo com Filho (2011), o contraste com um microcontrolador é extremamente parecido. Normalmente, microcontroladores possuem menos que 100K de RAM (apenas poucos KBytes), e freqüentemente são sistemas de chip único. Enquanto que o termo SoC é várias vezes usado para processadores mais potentes, capazes de executarem programas como o Windows ou o Linux, nos quais necessitam de memórias externas (flash, RAM) para funcionarem, e que são usados com vários periféricos acoplados. A grande maioria dos sistemas que se rotulam System-on-a-chip possuem uma conotação técnica maior do que a realidade: aumentam a integração do chip para reduzir os custos de fabricação e disponibilizar sistemas mais compactos. Muitos são complexos de mais para se ajustarem em apenas um chip construído com um processo otimizado, para apenas uma das funções do sistema.

A Figura 11, Texas (2005), ilustra o diagrama de blocos de um OMAP (*Open Multimedia Application Platform* – Plataforma Aberta de Aplicação multimídia) 2420, um *SoC* (*system-on-a-chip*) proprietário da companhia *Texas Instruments*, que combina um processador ARM e diversos outros controladores auxiliares em um único chip. Este modelo em particular é muito usado em *smatphones*, incluindo o modelo da Nokia N95.

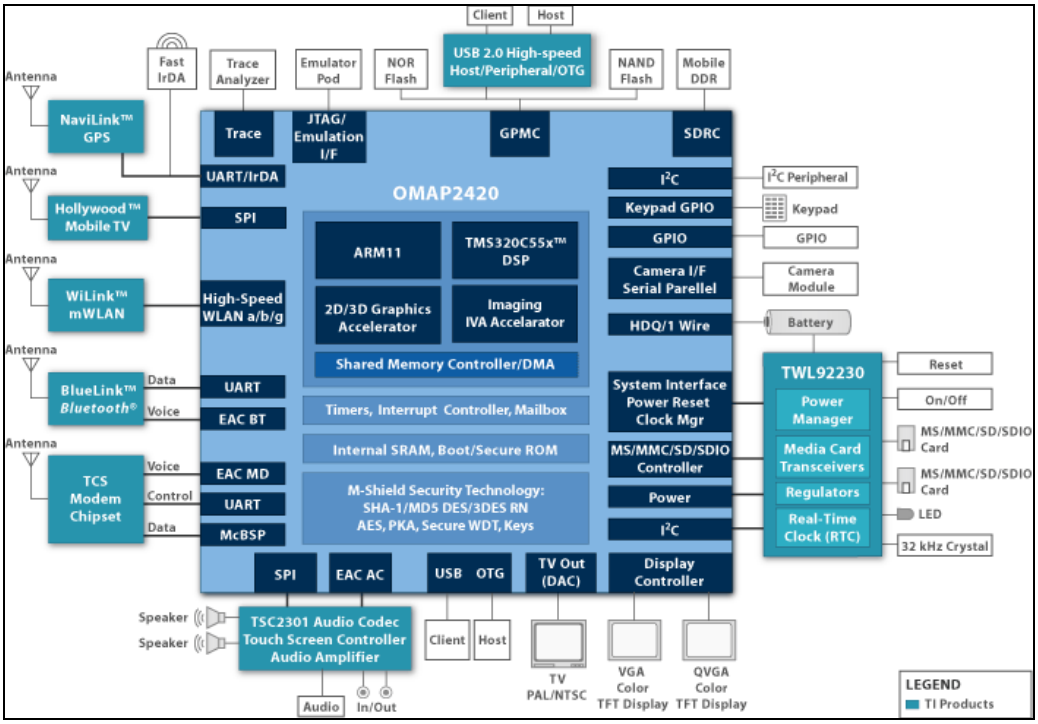

Figura 11: Diagrama de blocos do SoC TI OMAP2420

O OMAP2420 possui um acelerador de vídeo (2D), com a função de decodificar diversos formatos de arquivos, processar imagens e vídeos capturados pela câmera do *smartphone*, por exemplo, e também possui um acelerador de vídeo 3D dedicado, que é acionado quando são executados jogos ou outros aplicativos que utilizem gráficos 3D, Morimoto (2010).

Essa integração utilizada nos chips ARM proporciona um baixo consumo elétrico e uma redução no custo de produção, visto que ela é mais barata do que produzir vários chips separados.

2.6.2.1. O *TI* OMAP 3530

Nesta subseção será descrito o SoC no qual a placa utilizada para embarcar a SOA-DB se baseia, no caso a Beagleboard, que será discutida na subseção 2.6.2.2.

Segundo a Texas (2011), O TI OMAP 3530 (Figura 12) utiliza o super escalar ARM Cortex A8 (será discutido na próxima subseção), com opção para

a tecnologia de gerenciamento de energia dos CIs de forma discreta e integrada, denominada de *SmartReflex*.

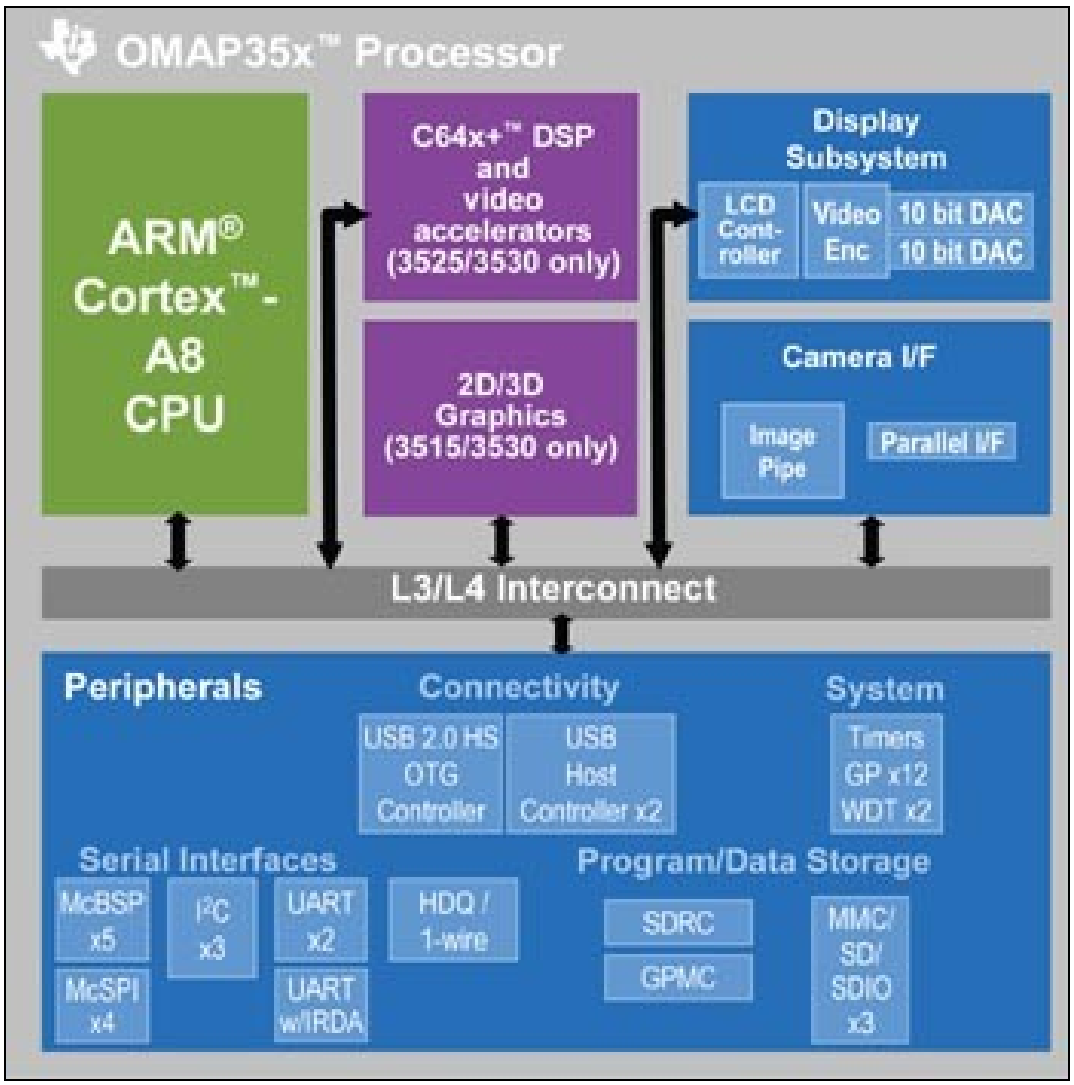

Figura 12: SoC *TI* OMAP 35x

O núcleo do Cortex A8 possui co-processador com tecnologia SIMD (*Single Instruction, Multiple Data* – Instrução Única, Múltiplos dados), com velocidades variando entre 600 a 720 MHz, e um DSP, o TMS320C64+. Para mais detalhes, consultar o guia do usuário do DSP, em Texas (2011-2).

No núcleo do Cortex A8, há também um motor de processamento gráfico 3D dedicado, o POWERVR SGX, com capacidade de processamento gráfico de até 10 milhões de polígonos por segundo. Para mais detalhes, consultar o site do fabricante em Imagination (2011).

Em relação às memórias do SoC, o processador principal possui 16KB para instruções mais 16KB para dados, totalizando 32KB de memória *cache* L1 e 256KB de memória *cache* L2. Já o DSP, possui 112KB de *cache* L1, sendo 32KB para programa, 32KB para dados e 48KB para uso geral. A *cache* L2 do DSP possui 64KB para programa e 32KB para dados, totalizando 96KB de memória SRAM, mais 16KB de memória ROM. Há ainda 64KB de memória SRAM e 112KB de memória ROM integrados no chip do SoC.

Em relação aos periféricos, o *TI* OMAP suporta as memórias de baixo consumo de energia LPDDR (*Low Power Double Data Rate* – DDR de baixo consumo). Suporta também as memórias *flash* NAND e NOR, SRAM e pseudo SRAM. Possui suporte à tecnologia USB 2.0, com dois controladores HOST. Para saída de vídeo, suporta interfaces LCD e TV (S-Vídeo), com PIP (*Picture-in-video*), conversão de espaço de cor, redimensionamento e rotação.

O SoC ainda possui outras interfaces seriais, como UART (Universal Asynchronous Receiver/Transmitter – Transmissor/Receptor Assíncrono Universal) e UART com IRDA (Infrared Data Association – Associação de dados infravermelho).

2.6.2.2. A Beagleboard

Segundo Beagleboard (2011), a Beagleboard (Figura 13) é uma placa de baixo custo baseada no SoC *TI* OMAP 3530, baseado em placa única, sem ventoinha, com desempenho similar a de um *laptop*, além de possuir medidas bem menores do que as de um microcomputador *desktop*, em torno de três polegadas de área, além de não possuir ruído de funcionamento, devido à ausência de ventoinha para o processador. A Tabela 1 ilustra as especificações da Beagleboard.

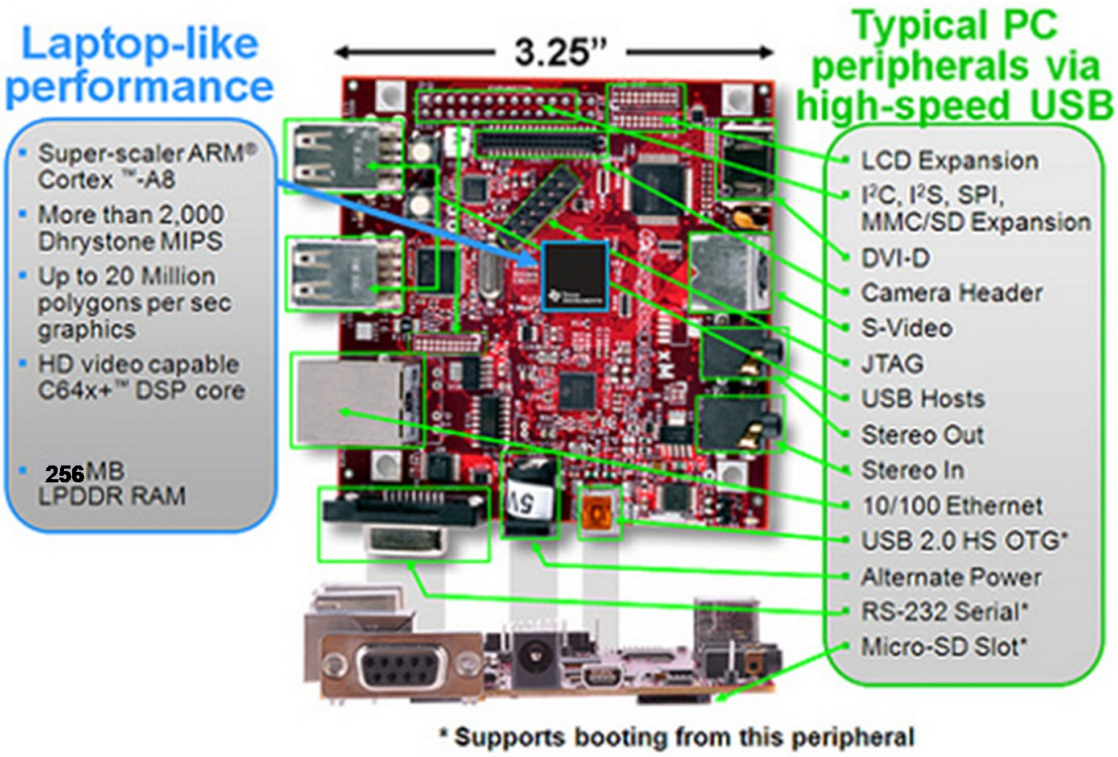

Figura 13: Visão frontal da Beagleboard.

Dentre as especificações da Tabela 1, podem ser destacadas:

- Em torno de 1200 DMIPS[1] fornecidos pelo Cortex A8, com previsão de desvios de alta precisão e 256KB de memória cache L2, rodando a 600MHz, com possibilidade de overclock até 720MHz.
- Suporte à aceleração gráfica 2D/3D, com o OpenGL ES 2.0, com capacidade de renderização de 20 milhões de polígonos por segundo.
- Capacidade de executar vídeos em alta definição, através do DSP TMS320C64x+, chegando até 430 MHz.
- Alimentação completa através do chipset USB com uma lógica de mínimo consumo de energia adicional.
- Suporte à conexão de monitores DVI-D.

[1] Medição de desempenho, Weicker (1984), significando Dhrystone MIPS, obtido quando o índice Dhrystone é dividido por 1757 (o número de Dhrystones por segundo, obtido na VAX 11/780, uma máquina de 1 MIPS nominal).

- Compatibilidade com uma série de periféricos USB, incluindo HUBS, teclados, mouses, WiFi, Bluetooth, webcams, entre outros.
- Interface de cartões de memória MMC+/SD/SDIO.
- Saída S-Vídeo para conexões com televisões NTSC e PAL ou visores externos.
- Entrada e saída de Áudio estéreo para microfones e fones de ouvido.
- Alimentação através de carregadores USB típicos, como os de laptops, adaptadores automotivos, baterias ou carregadores solares.
- Suporta sistemas operacionais específicos para a arquitetura ARM, como o Angstrom, Ubuntu (ARM), Android, Debian (ARM) e GeeXboX ARM.

Tabela 1: Especificações da Beagleboard

	Feature	
Processor	OMAP3530DCBB72 720MHz	
POP Memory	Micron	
	2Gb NAND (256MB)	2Gb MDDR SDRAM (256MB)
PMIC TPS65950	Power Regulators	
	Audio CODEC	
	Reset	
	USB OTG PHY	
Debug Support	14-pin JTAG	GPIO Pins
	UART	LEDs
PCB	3.1" x 3.0" (78.74 x 76.2mm)	6 layers
Indicators	Power	2-User Controllable
	PMU	
HS USB 2.0 OTG Port	Mini AB USB connector	
	TPS65950 I/F	
	MiniAB	
HS USB Host Port	Single USB HS Port	Up to 500ma Power
Audio Connectors	3.5mm	3.5mm
	L+R out	L+R Stereo In
SD/MMC Connector	6 in 1 SD/MMC/SDIO	4/8 bit support, Dual voltage
User Interface	1-User defined button	Reset Button
Video	DVI-D	S-Video
Power Connector	USB Power	DC Power
Expansion Connector (Not Populated)	Power (5V & 1.8V)	UART
	McBSP	McSPI
	I2C	GPIO
	MMC	PWM
2 LCD Connectors	Access to all of the LCD control signals plus I2C	3.3V, 5V, 1.8V

2.6.2.3. Arduino

Arduino (2011) é uma plataforma de prototipagem eletrônica, baseada em código com um microcontrolador baseado em placa única, descendente da plataforma de código aberto *Wiring* (2011), designada a tornar o processo de multidisciplinaridade na eletrônica mais acessível. O Hardware consiste de um hardware simples de código aberto designado para a placa Arduino, com um processador Atmel (2011) AVR e uma interface I/O *on-board*. O software consiste em um compilador de linguagem de programação padrão e do carregador de boot que é executado na placa. A plataforma Arduino é indicada para artistas, projetistas, entusiastas e qualquer um que esteja interessando em criar objetos e ambientes interativos. A Figura 14 ilustra a placa Arduino Uno, a versão mais utilizada desta plataforma, equipada com o microcontrolador ATmega328P, de 8bits, rodando a 16MHz.

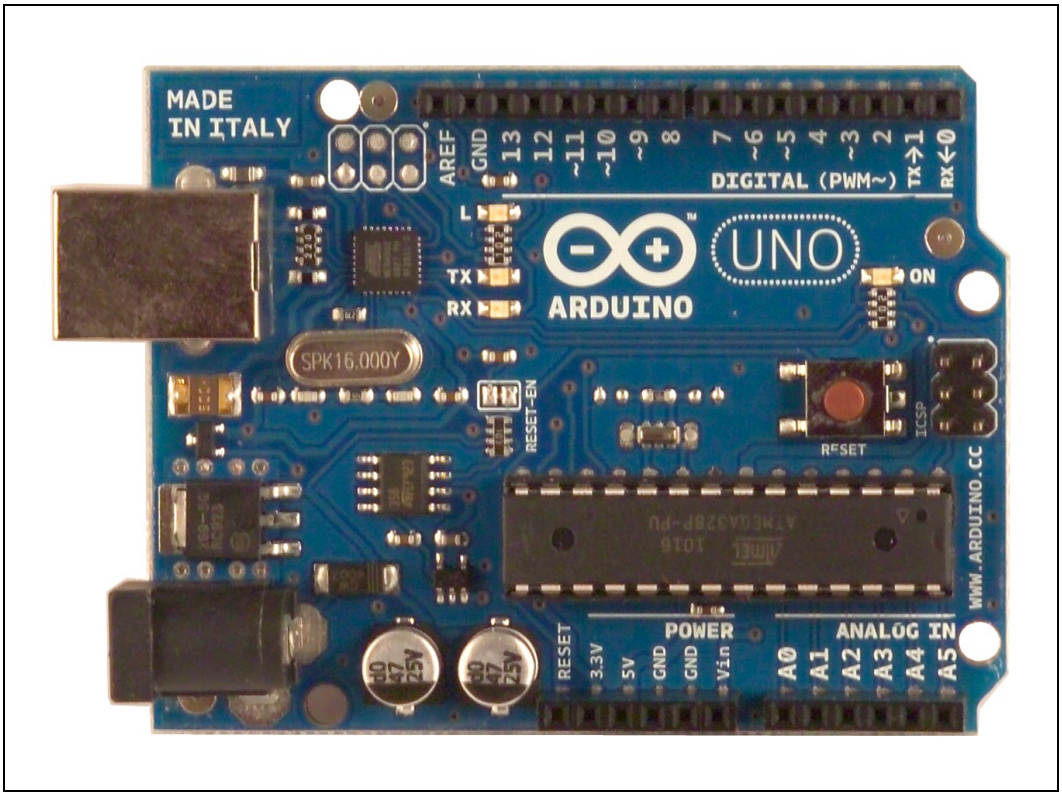

Figura 14: Visão frontal da placa Arduino Uno.

O hardware do Arduino é programado utilizando uma linguagem baseada em *Wiring* (sintaxe e bibliotecas), similar a C++, com algumas

modificações e simplificações, junto com uma IDE baseada em *Processing* (2011).

As placas podem ser construídas manualmente ou compradas pré-montadas e o software pode ser baixado gratuitamente. Os projetos de hardware de referência (arquivos CAD) estão disponíveis sob uma licença open-source, podendo ser adaptados de acordo com as necessidades.

A placa Arduino UNO suporta dois tipos de alimentação, via fonte de 5V ou via entrada USB fêmea. Quanto à comunicação, ela possui seis entradas analógicas e 14 pinos digitais (via interface RS-232), podendo ser utilizados para entrada ou saída de dados. Há também um hardware interno, conversor serial-USB, que possibilita a ligação da placa em portas USBs convencionais, porém a comunicação é toda realizada via RS-232.

2.7. Java Embedded SE

Segundo a ORACLE (2010), a Java SE Embedded deriva do Java SE. Ela suporta as mesmas plataformas e funcionalidades do Java SE, provê qualidades específicas e como seu próprio nome sugere, suporta o mercado embarcado. Esse suporte e as qualidades específicas para dispositivos embarcados incluem plataformas adicionais (arquiteturas ARM e Power), como pequenos espaços ocupados pelas JREs, configurações mais leves e otimizações de memória.

Atualmente, ORACLE (2010), a versão da máquina virtual Java, otimizada para dispositivos embarcados, está na versão 6u21 e representa um passo importante na sincronização da JRE para as plataformas de Linux embarcado em ARM, arquitetura Power e x86. As mudanças dessa versão da Java SE Embedded comparadas às anteriores dependem da plataforma. Para a arquitetura Power e x86, a Java SE Embedded anterior foi baseada na Java SE 5u10, já a versão para processadores ARM foi baseada no Java SE 6u21.

O desenvolvimento de aplicativos com a utilização da Oracle Java Embedded SE é livre, porém *royalties* são requeridos caso seja desenvolvido um sistema com essa tecnologia. Para maiores detalhes, consultar a licença em ORACLE (2011). De qualquer forma, esses produtos são totalmente

compatíveis com a Java SE, que proporciona o reuso de qualquer código Java de outras plataformas ou produtos, sem ter que passar por ciclos de portabilidade, recodificação e testes.

2.7.1. Requisitos de sistema

As Tabela 3, Oracle (2010), ilustra os requisitos necessários para a execução do ambiente de desenvolvimento Java *Embedded* SE nas plataforma ARM. Além da ARM, são suportadas a Arquitetura Power e x86, específicas para sistemas embarcados.

A propriedade *Headful* se refere à possibilidade do sistema ser controlado por um mouse e teclado e ter interface gráfica. Apenas a plataforma "pura" ARMv7 possui essa propriedade, suportando a partir da versão X11R6, que é uma interface gráfica típica de ambientes UNIX. Para dispositivos mais simples que esses da Tabela 2, como por exemplo, um dispositivo que não consiga fornecer 32MB de memória RAM exclusivamente para a máquina virtual Java, a Oracle (2010) recomenda a Java ME (*Micro Edition*).

Tabela 2 – Requisitos de sistema da Java Embedded SE para as plataformas ARM

Java SE for Embedded 6 on ARM EABI, Little-Endian, Linux			
Arquitetura	ARM v5	ARM v6/v7	ARM v7
Sistema Operacional	Linux: kernel 2.6.28 ou mais atual; glibc 2.9 ou superior	Linux: kernel 2.6.28 ou mais atual; glibc 2.9 ou superior	Linux: kernel 2.6.28 ou mais atual; glibc 2.9 ou superior
Ponto flutuante	Soft Float	Hard Float (VFP)	Hard Float (VFP)
Headful	Não	Não	Sim. X11R6 ou superior
RAM	32MB ou mais para a Java	32MB ou mais para a Java	64MB ou mais para a Java
ROM/Flash/Disk	37MB ou mais para a Java	37MB ou mais para a Java	46MB ou mais para a Java
Java SE for Embedded version	6 Update 21 (B09)	6 Update 21 (B09)	6 Update 21 (B09)

Capítulo 3

SOA-DB

No cenário atual há várias otimizações do funcionamento de equipamentos hospitalares, juntamente com suas respectivas soluções de software, seja elas concentradas num ambiente de software distribuído, Lacerda (2010), seja embarcado no equipamento médico, De Capua (2010), ou até mesmo soluções híbridas, Wang (2007).

Essas soluções incluem um monitoramento remoto de pacientes de maneira eficiente, independência de plataforma da aplicação cliente, um pequeno espaço físico ocupado e abstração no protocolo de acesso ao dispositivo biomédico. O problema é que nenhuma dessas soluções fornece todas essas qualidades juntas.

Levando esse problema em consideração, a proposta da SOA-DB é realizar o monitoramento remoto de pacientes de forma eficiente, com abstração do protocolo de acesso ao dispositivo biomédico e independência de plataforma por parte da aplicação cliente que acessa o dispositivo, utilizando um dispositivo fisicamente pequeno, de baixo consumo elétrico e escalável, de forma que a arquitetura se adapte a mudanças futuras no protocolo de acesso ao dispositivo biomédico, ou seja, tenha a capacidade de ser escalável.

3.1. SOA-DB no ambiente hospitalar

A Figura 15 ilustra o SOA-DB inserido num ambiente hospitalar extremamente heterogêneo, tanto nas aplicações que acessam os dispositivos biomédicos, quanto nos próprios dispositivos, ou seja, o SOA-DB age como elemento de acesso entre as aplicações e os dispositivos, deste modo, tornando homogêneo o acesso a esses.

Desenvolvedores de aplicações que utilizam linguagens de programação como Java, C# .NET ou Python, ou qualquer outra que forneça suporte a SOA, poderão acessar dispositivos de hardware, sem a necessidade de implementar o acesso a esses.

Nesse ambiente, as requisições são realizadas por diferentes aplicações clientes de forma remota, através de uma arquitetura orientada a serviço, que na SOA-DB é implementada através de Web Services. Basicamente, são requisições realizadas na linguagem de programação nativa da aplicação, que necessita acessar um dispositivo qualquer.

Além de diferentes aplicações clientes, a SOA-DB também pode acessar dispositivos médicos que implementem diferentes interfaces de acesso físico, tanto as interfaces comumente difundidas no mercado, como RS-232 ou USB (*Universal Serial Bus* – Barramento Serial Universal) quanto protocolos de comunicação específicos do meio hospitalar, como DICOM exposto em Noumeir (2003) ou HL7 descrito em Eggebraaten et. al. (2007).

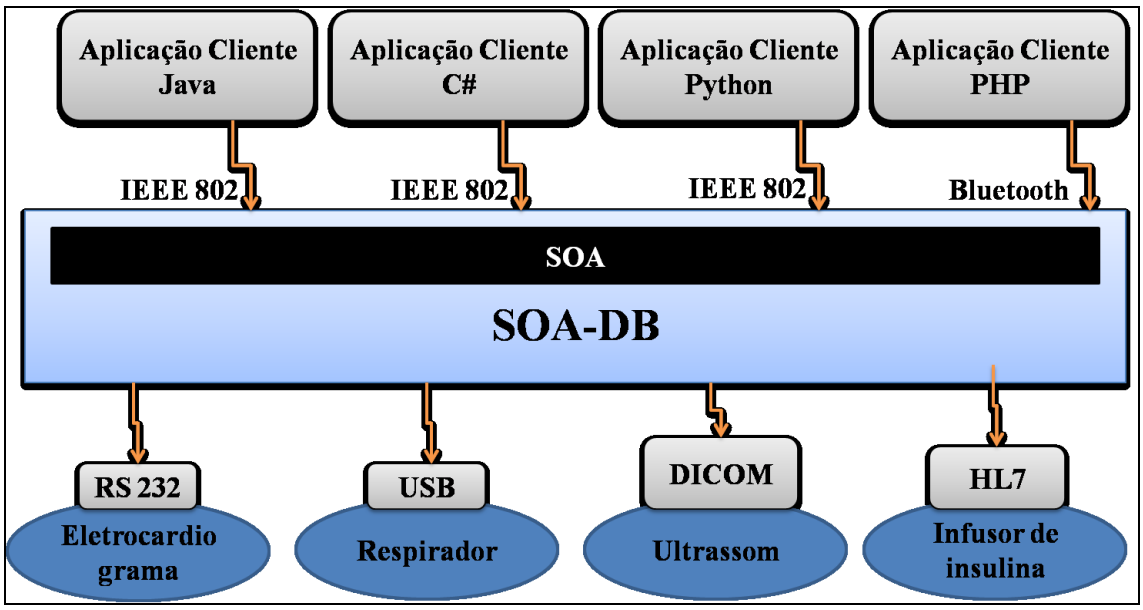

Figura 15: SOA-DB no ambiente hospitalar.

Essas requisições operam sobre o padrão W3C XML. O código XML que chega ao SOA-DB é processado e, com base nesse processamento, a SOA-DB realiza o acesso ao dispositivo de destino desejado.

3.2. Arquitetura do Sistema

A SOA-DB (Figura 16) é composta por dois componentes principais: A implementação das interfaces de comunicação (API Mult-I/O) e a implementação de um serviço baseado em SOA (Componente SOA).

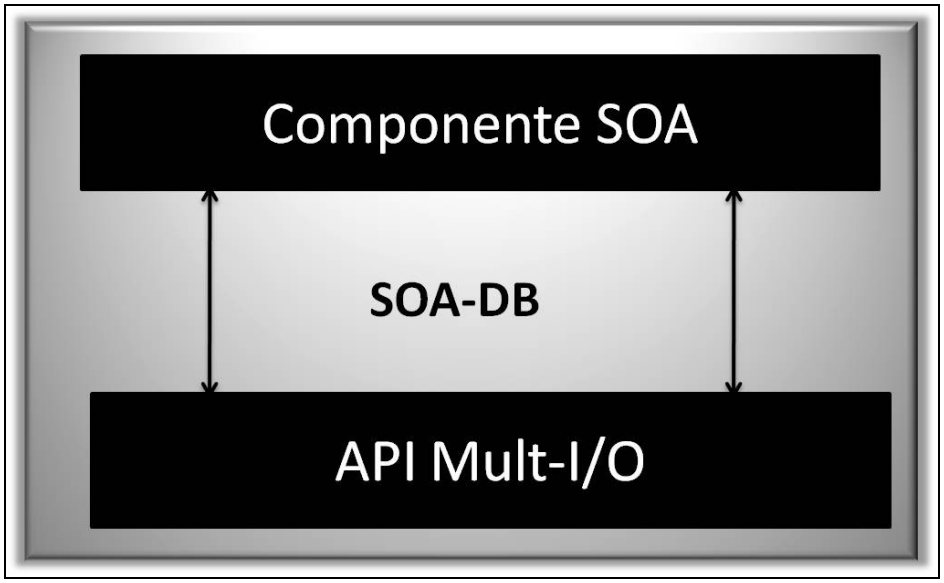

Figura 16: Arquitetura da SOA-DB.

A API Mult-I/O (Figura 17) integra as diferentes implementações de acesso a dispositivos, tais como USB, Serial, Paralela, IEEE 802. Essa API provê uma interface remota para a aplicação que servirá de base para as requisições de I/O. As diferentes implementações são definidas de acordo com a interface de comunicação que o dispositivo implementa.

Durante o processo que estabelece a comunicação, a arquitetura SOA-DB utiliza os padrões de projeto *Abstract Factory* e *Factory Method* para criar uma interface adequada ao dispositivo. Para isso é realizada uma consulta ao Repositório de Dispositivos, que contém as informações pertinentes aos dispositivos integrados à SOA-DB. A principal função da API Mult-I/O é abstrair o protocolo de comunicação de acesso ao dispositivo biomédico.

A Figura 19 ilustra o Repositório de Dispositivos implementado na SOA-DB, que pode ser comparado ao "*Device Service Registry*" do padrão SODA descrito na Figura 1. Este repositório funciona como um Serviço de Nomes,

podendo ser implementado em um arquivo texto, um arquivo XML, um Sistema de Gerenciamento de Banco de Dados (SGBD) ou qualquer outra forma de armazenamento de dados. Esse repositório possui duas tabelas: INTERFACE e DISPOSITIVO, de modo que o relacionamento entre elas é, respectivamente, de um para muitos.

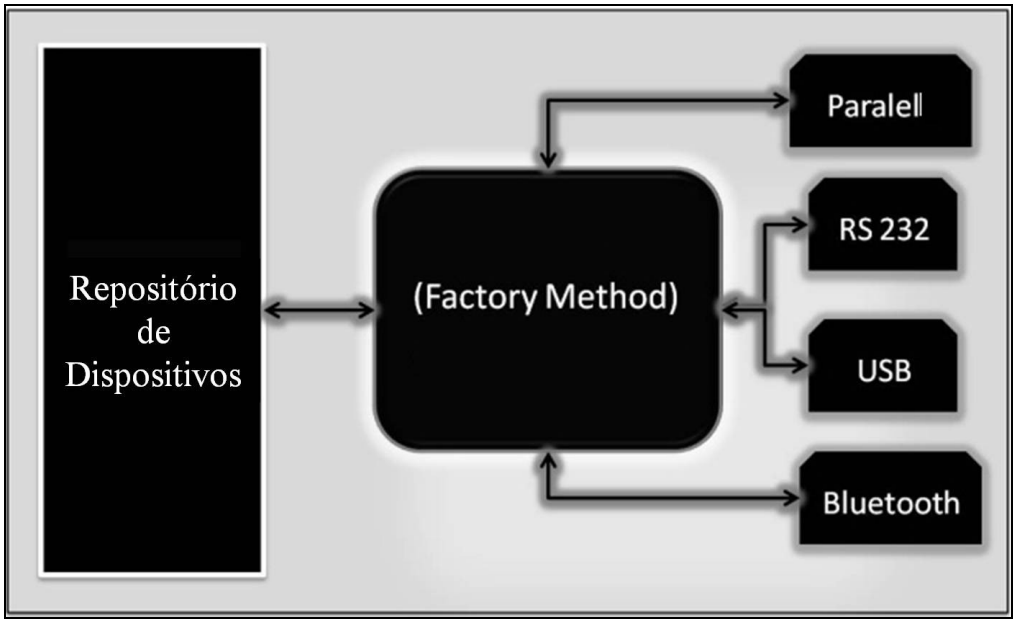

Figura 17: A API Mult-I/O.

O componente SOA (Figura 20) terá como base o framework SOAP, que é a implementação do protocolo SOAP direcionada a certa linguagem de programação, ou seja, ele sofrerá variação de acordo com a linguagem implementada. Por exemplo, o Apache Axis para a linguagem Java ou Nusoap para a linguagem PHP. O protocolo SOAP é responsável pelo transporte das requisições e respostas (mensagens em XML) pela rede.

O padrão W3C WSDL descreve em serviços os recursos disponibilizados pelo dispositivo. Ele servirá de base para a Aplicação Cliente consumir os serviços disponibilizados pelos dispositivos médicos de maneira satisfatória. A Figura 18 exemplifica o WSDL no contexto da SOA-DB, implementado em XML.

```
<element name="lerDispositivo">
    <complexType>
     <sequence>
      <element name="dispositivo" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="lerDispositivoResponse">
    <complexType/>
   </element>
   <element name="lerDados">
    <complexType/>
   </element>
   <element name="lerDadosResponse">
    <complexType>
     <sequence>
      <element name="lerDadosReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
<element name="escreverDispositivo">
    <complexType>
     <sequence>
      <element name="msg" type="xsd:string"/>
      <element name="dispositivo" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="escreverDispositivoResponse">
    <complexType/>
   </element>
```

Figura 18: Trecho do WSDL no contexto da SOA-DB

O trecho do WSDL da Figura 18 é a interface na qual as aplicações clientes dos dispositivos implementarão para acessá-los de forma transparente. Esse exemplo mostra duas operações disponibilizadas: A leitura (“lerDispositivo”) de dados do dispositivo e a escrita (“escreverDispositivo”) no dispositivo.

Portanto, o Componente SOA fornece a modelagem de um dispositivo biomédico como um serviço remoto.

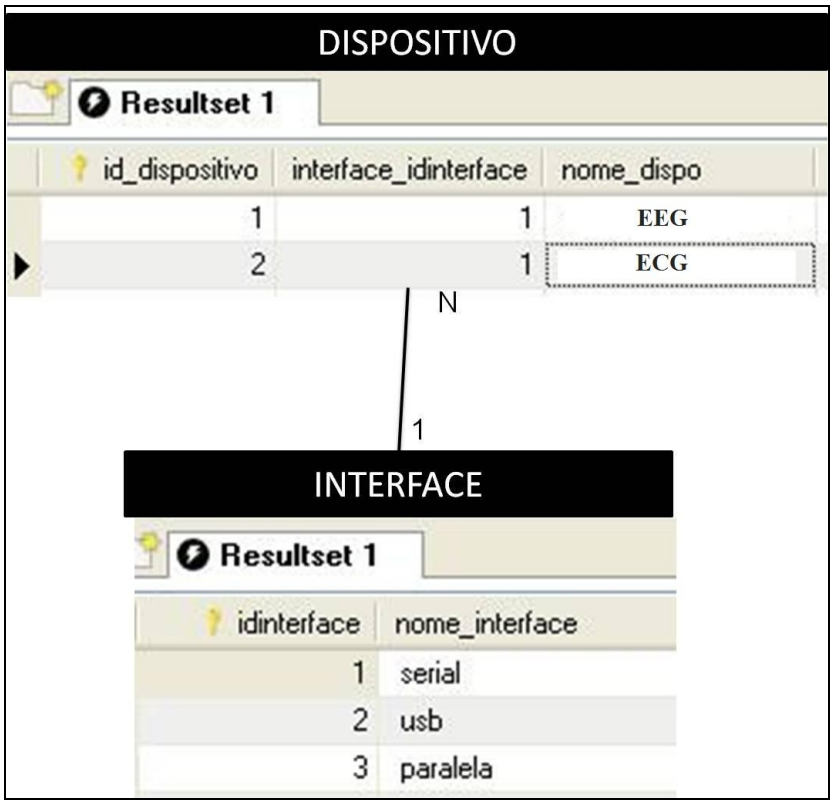

Figura 19: O Repositório de Dispositivos.

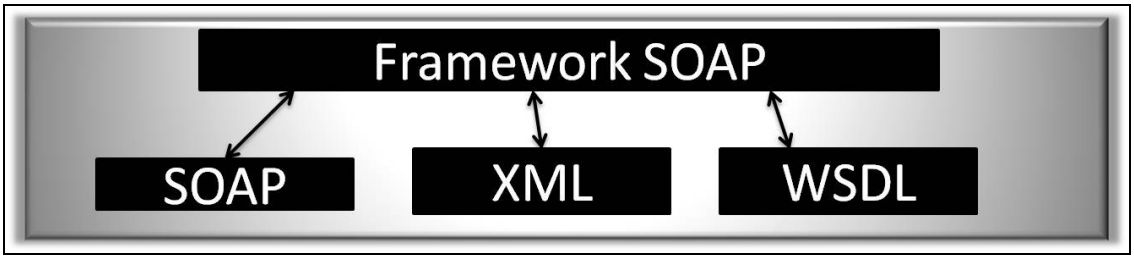

Figura 20: O Componente SOA.

3.3. Tratamento de requisições à SOA-DB

No contexto deste trabalho, uma requisição é considerada como o processo de solicitação de leitura ou escrita em um dispositivo biomédico, por meio da SOA-DB. Como um dos requisitos da SOA-DB é a possibilidade de acesso aos dispositivos por mais de um cliente, foi preciso elaborar uma estratégia para tratar essas múltiplas requisições.

Esse ambiente é ilustrado na Figura 21, onde o processo de Requisição inicia-se com as Aplicações Clientes requisitando dados aos dispositivos registrados na SOA-DB.

O primeiro Cliente que acessar um determinado dispositivo (isto será definido automaticamente pelo servidor WEB instalado na SOA-DB) ativará o procedimento remoto referente àquela requisição, mas o dado não será retornado diretamente para o Cliente. Esse dado é inicialmente inserido em uma tabela temporária de um SGBD (Sistema Gerenciador de Banco de Dados), que será totalmente atualizada, quando possuir um número x de dados, por questões de desempenho do dispositivo que hospeda a arquitetura.

Posteriormente, o dado que efetivamente retornará ao Cliente, como resposta de sua requisição, é o dado mais antigo da tabela temporária, garantindo assim, que os dados sejam acessados pelo Cliente de maneira íntegra.

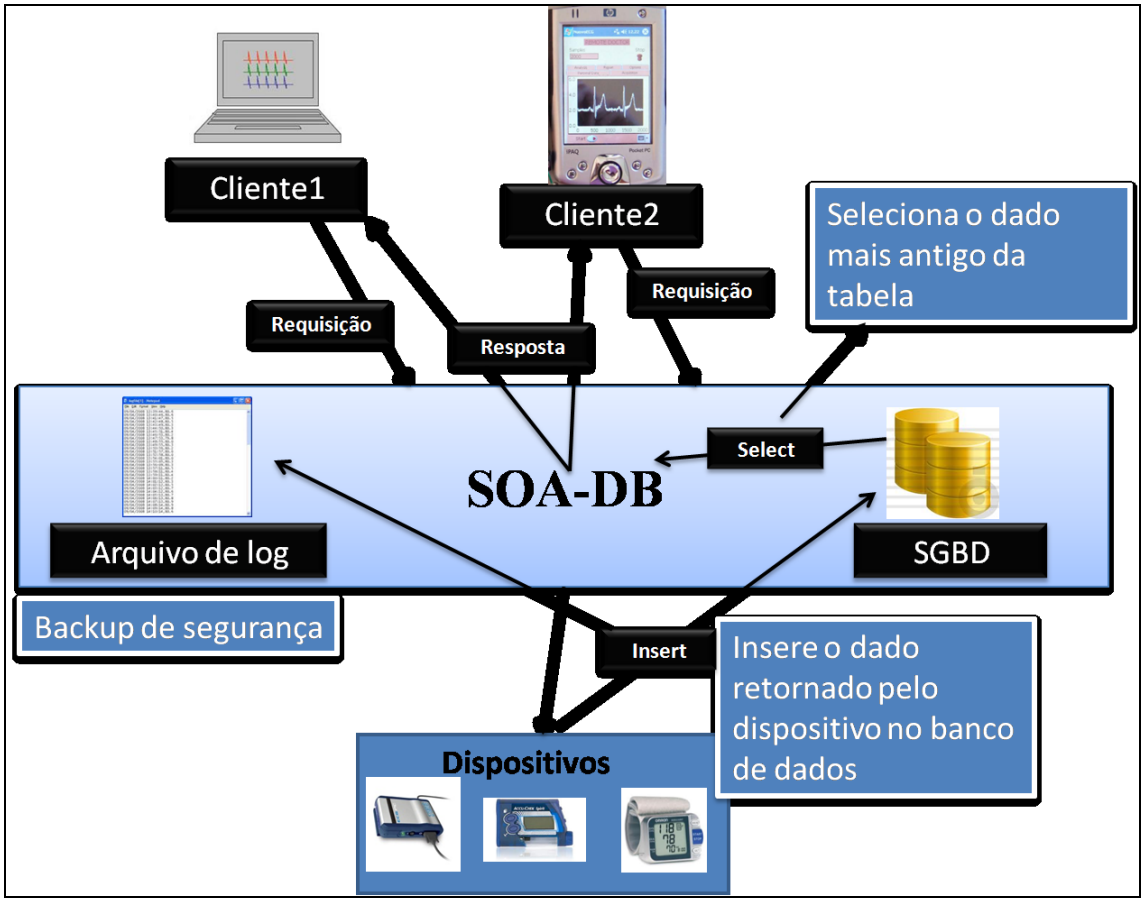

Figura 21: Tratamento de requisições para múltiplos clientes

Adicionalmente, existe um arquivo de log., que armazena todos os dados inseridos na tabela temporária. Em caso de falhas no banco de dados, esse arquivo de log. é utilizado como *backup* de segurança.

3.4. Comparativo entre SOA-DB e SODA

A arquitetura da SOA-DB foi amplamente baseada nos conceitos de SODA propostos por Deudg et. al. (2006). A Figura 1 apresentada na subseção 2.1 será repetida nesta subseção (Figura 22), com o intuito de visualizar mais facilmente como a SOA-DB implementa os conceitos de SODA.

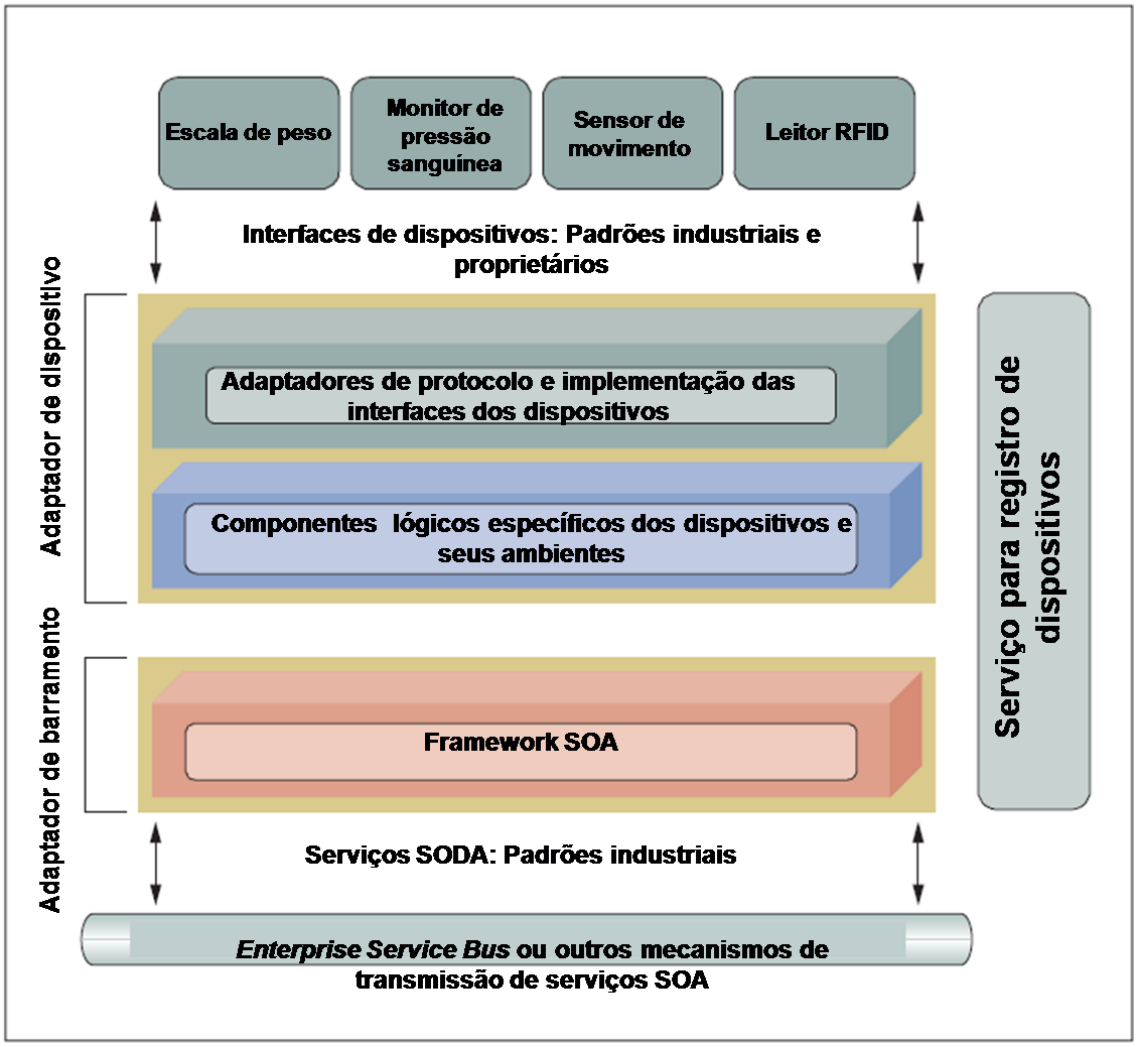

Figura 22: Descrição dos componentes de SODA

Portanto, um comparativo com os componentes descritos da SOA-DB, na subseção 3.2, pode ser feito com os componentes de SODA. A Figura 23 estabelece esse comparativo, tomando como base os conceitos abstratos de SODA e aplicando os componentes concretos da SOA-DB.

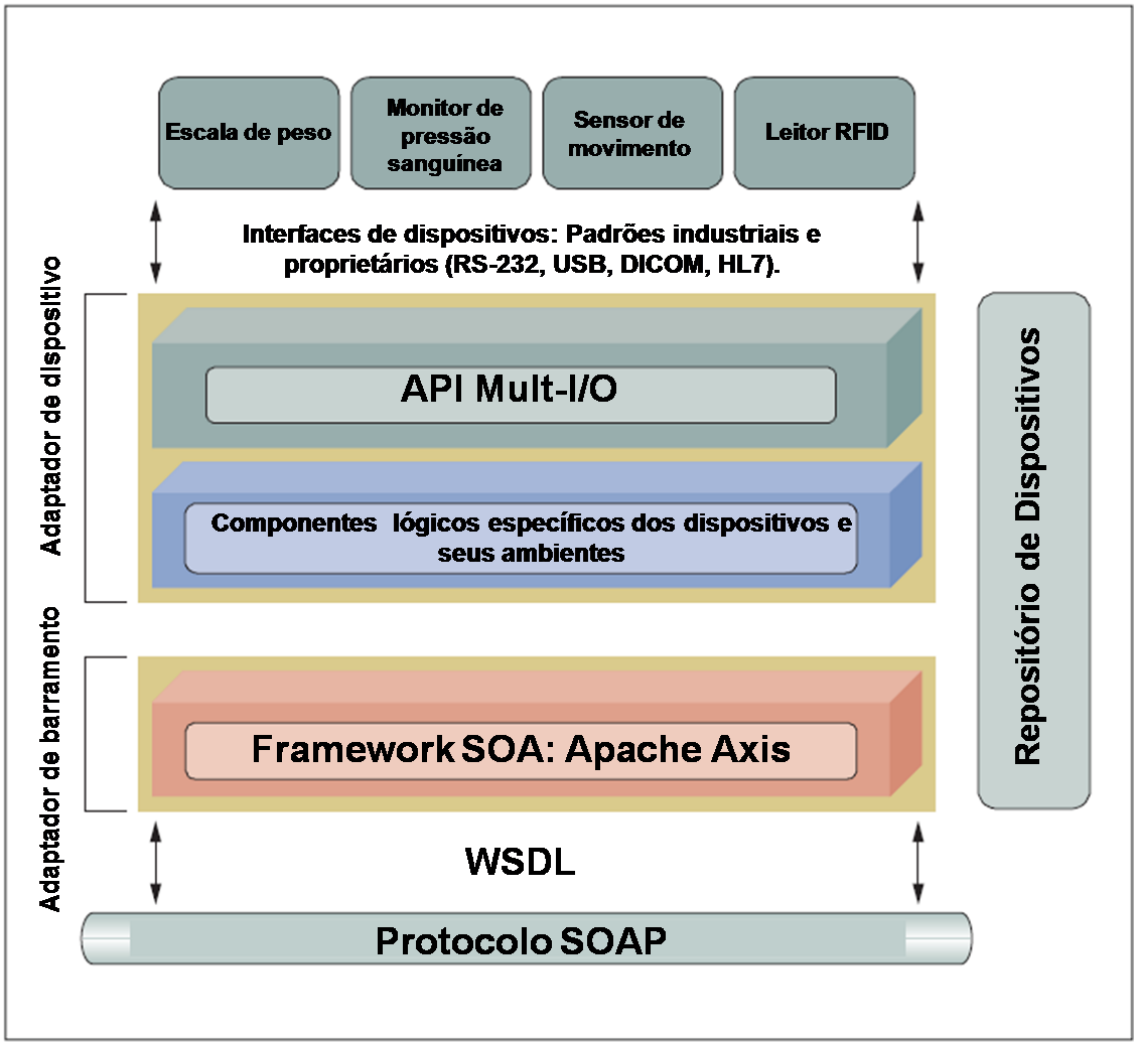

Figura 23: SOA-DB como implementação de SODA

Como conseqüência, esse comparativo serve para reforçar a viabilidade teórica da SOA-DB, já que esta é baseada em conceitos amplamente difundidos na literatura.

3.5. SOA-DB embarcada

Levando em consideração o conceito de mobilidade, surgiu a necessidade de embarcar a arquitetura. Para tanto, alguns pré-requisitos foram considerados:

- Dispositivo de pequeno porte (em torno de 3 polegadas de área);
- Suportar comunicação móvel;
- Baixo custo financeiro;

- Baixo consumo energético;
- Suportar a versão SE do Java;
- Suportar um contêiner WEB Java.

Sem os dois últimos pré-requisitos, seria preciso alterar de maneira significativa a estrutura da SOA-DB, pois se pensando em dispositivos embarcados aliados à linguagem de programação Java, a solução recorrente é a utilização da versão *Micro Edition (ME)* da Java, própria para dispositivos de baixo poder de processamento.

Porém, para se ter o suporte à Java SE e um contêiner Web Java, é preciso ter um hardware com um razoável poder de processamento e capacidade de memória RAM. O avanço tecnológico, tornou esses requisitos possíveis, através dos *SoCs* citados na subseção 2.6.1.

O SoC escolhido foi a *BeagleBoard*, baseado no SoC TI OMAP 3530, com um processador ARM Cortex A8 de 600 MHz e 256 MB de memória RAM. A partir dessa configuração de hardware, com a adição de um sistema operacional UNIX específico e uma versão da máquina virtual Java, ambos para sistemas embarcados (*Angstrom e Java Emdedded SE* respectivamente), tornou-se viável a opção de embarcar a SOA-DB.

A Figura 24 ilustra um ambiente hipotético de operação da SOA-DB.

A partir do circuito de um dispositivo biomédico, no caso hipotético da Figura 24, um eletrocardiógrafo, uma placa de fenolite é impressa, finalizando a confecção do dispositivo biomédico. Os canais analógicos desse dispositivo são conectados num conversor A/D.

O conversor A/D escolhido foi uma placa eletrônica de código aberto, contendo um microcontrolador para realizar essa conversão. Após a conversão, os sinais já digitalizados serão enviados para a *Beagleboard,* que contém a SOA-DB embarcada. A SOA-DB recebe esses sinais e os converte para XML. Após isso, esses dados podem ser armazenados na própria placa ou acessados por uma aplicação cliente de monitoramento médico.

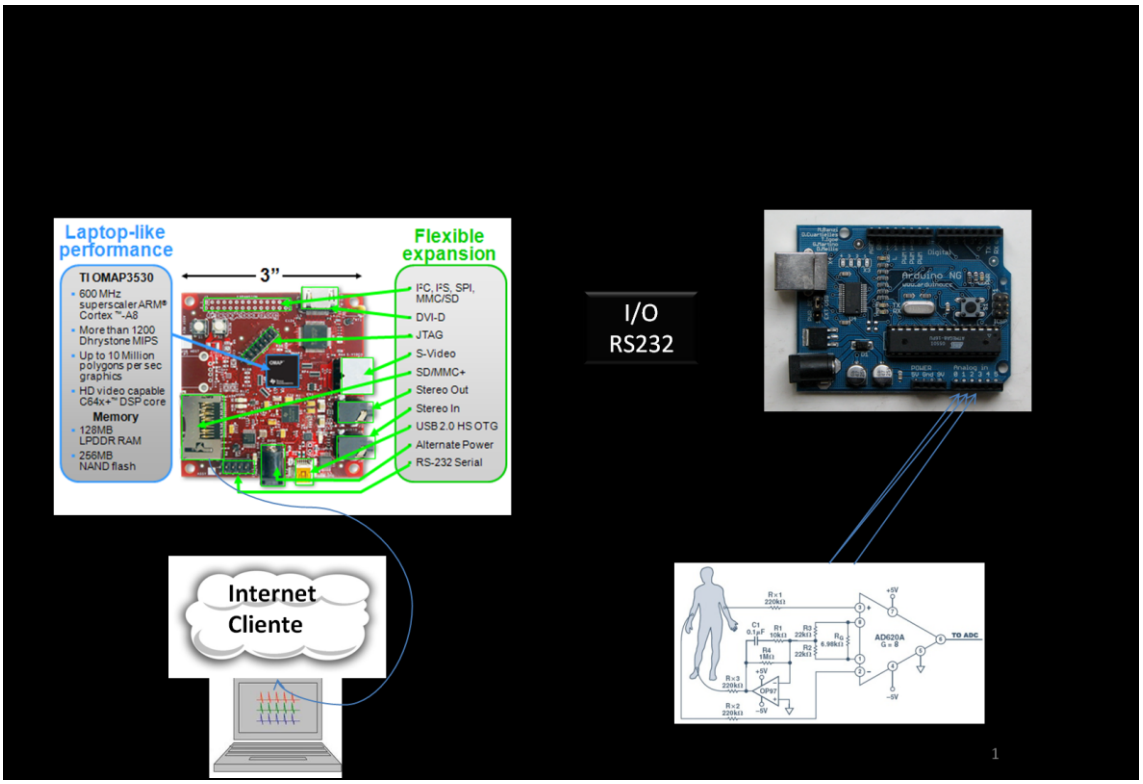

Figura 24: A SOA-DB embarcada

3.6. Ambientes de testes

O objetivo dos testes foi realizar medidas de desempenho da SOA-DB em ambientes diversificados, com o intuito de demonstrar a sua viabilidade, independente da arquitetura de hardware e software onde ela seja implantada.

Foram utilizados 3 (três) cenários para a realização dos testes. No primeiro cenário, a SOA-DB foi implantada em uma arquitetura de hardware x86, com dados simulados, posteriormente com dados emulados e por último, numa arquitetura ARM com um dispositivo real.

3.6.1. Ambiente 1 - SOA-DB na arquitetura x86 (Dados simulados)

- Configurações de Hardware
    - 03 microcomputadores desktop (PC01, PC02 e PC04)
        - Configuração: Intel Pentium 4 HT, 3.2 MHz, com 1GB de memória RAM;

- 01 microcomputador notebook (PC03)
  - Configuração: AMD Turion II Mobile, 1.6 MHz, com 2GB de memória RAM;
- 01 cabo de comunicação serial DB 9 null modem (fêmea-fêmea).

- Configuração de Software:
  - Sistema operacional Windows para os 4 (quatro) PCs;
  - Linguagem de programação Java para o desenvolvimento da SOA-DB;
  - Linguagens Java e C# para implementação dos clientes da SOA-DB;
  - API Java RXTX Serial: Utilizada para prover a comunicação física, RS-232, entre os dispositivos e a SOA-DB;
  - Apache Axis: Utilizado para implementar o padrão SOA;
  - Sistema de Gerenciamento de Banco de Dados MySQL.

Para este ambiente foram realizados testes na ausência e presença da SOA-DB, com o intuito de calcular o custo da adição da camada SOA-DB no acesso a dispositivos. A Figura 25 ilustra o acesso aos dispositivos realizados da maneira habitual. Nessa modalidade (sem a SOA-DB), os PC01 e PC02 simularam os dispositivos biomédicos, conectados ao PC03, por meio de um cabo RS-232 DB 9 null modem (fêmea-fêmea), a uma velocidade de 9600 bps.

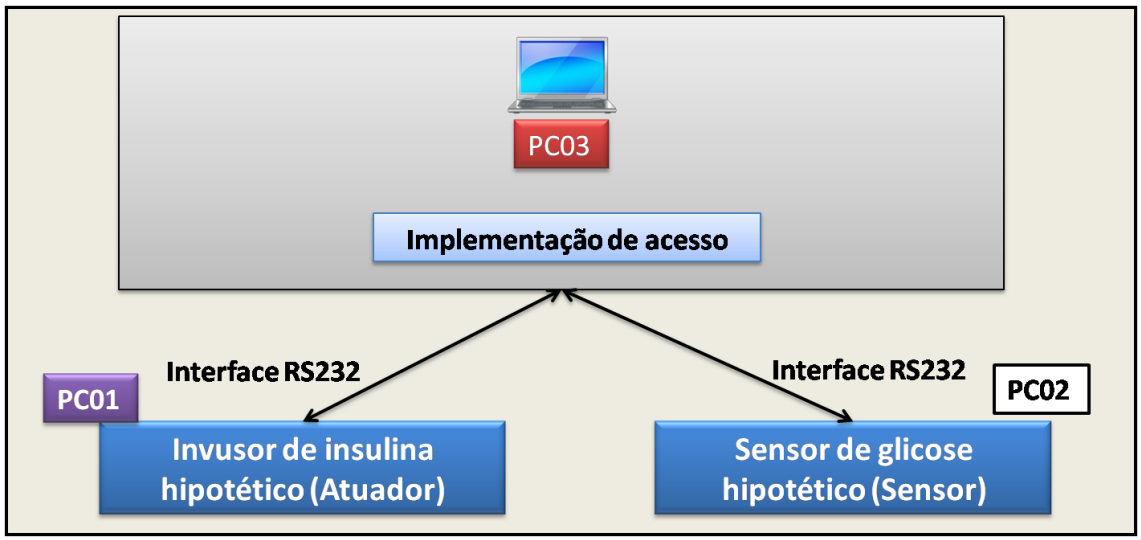

Figura 25: Acesso aos dispositivos da maneira habitual

Já a Figura 26 ilustra o ambiente com a adição da SOA-DB. Nesse ambiente, os dispositivos biomédicos foram acessados por duas aplicações clientes distintas, como clientes de *Web Service*, desenvolvidas nas linguagens Java e C# .NET (PC04) – de modo a verificar a capacidade de acesso por aplicações heterogêneas. As aplicações clientes do PC04 realizaram suas conexões aos PCs 01 e 02, que simularam os dispositivos biomédicos, por meio do PC 03 que continha a SOA-DB, deste modo, abstraindo dos clientes a comunicações no protocolo RS-232 de mais baixo nível. Por fim, os PCs 03 e 04 (aplicações clientes Web Services e a SOA-DB) foram interligados via interface de rede no padrão IEEE 802.03 (Switch Ethernet) a 100Mbps.

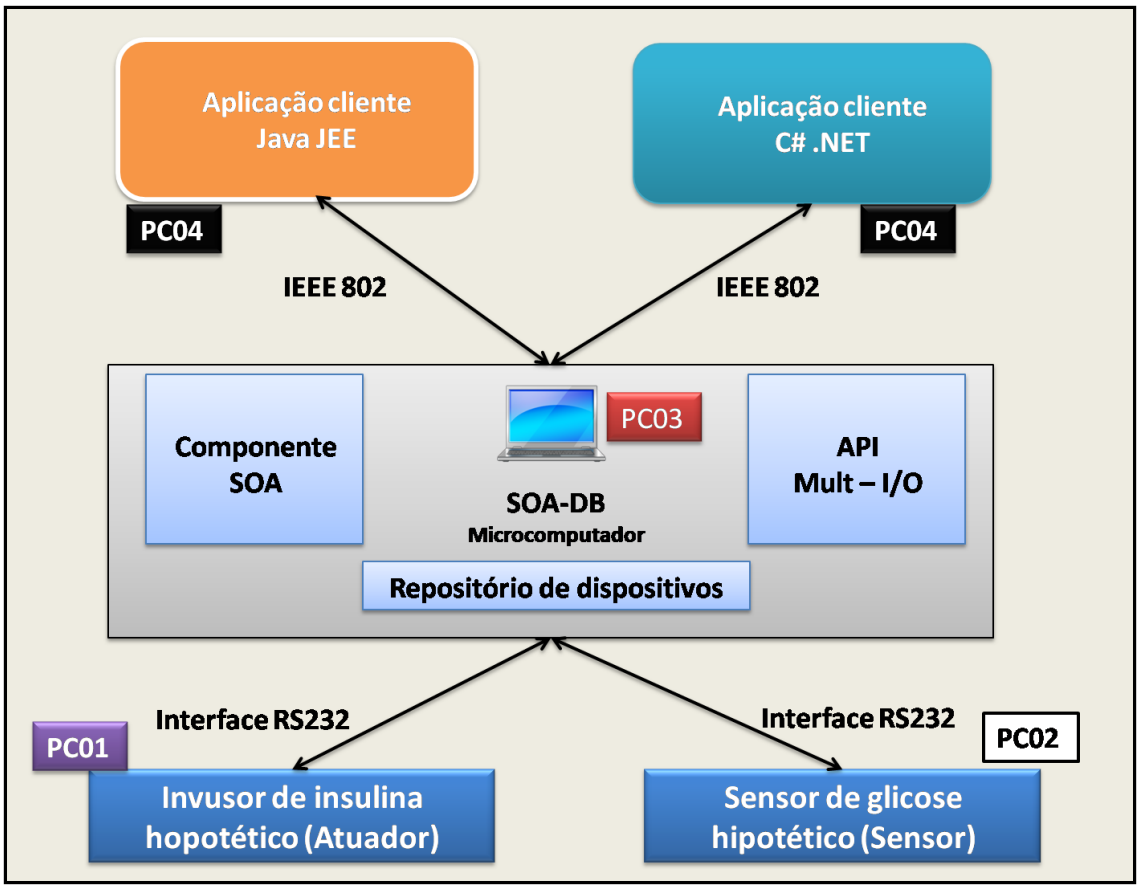

Figura 26: Ambiente de testes com dados simulados utilizado pela SOA-DB

3.6.1.1. Resultados experimentais

A Tabela 3 ilustra os resultados referentes ao ambiente ilustrado na Figura 25, sendo os dispositivos hipotéticos acessados (PC 01 e 02) por duas aplicações de leitura e escrita, respectivamente, na porta serial dos dispositivos biomédicos. O Tempo de acesso é o tempo de processamento da aplicação que acessa o dispositivo. Foram utilizadas 100 acessos aos dispositivos, obtendo-se o tempo médio de acesso para cada dispositivo.

Tabela 3: Acesso convencional aos dispositivos biomédicos

Dispositivo	Tempo médio de acesso (Milissegundos)
Infusor de insulina (atuador)	**124**
Sensor de glicose (sensor)	**1030**

Os resultados apresentados nas Figuras 27, 28, 29 e 30 são provenientes dos experimentos realizados com a utilização do SOA-DB, por meio das duas aplicações clientes da Figura 26: Java e C# .NET.

Neste contexto, faz-se necessário, antes de apresentar os resultados, definir os tópicos trabalhados nos experimentos: Requisição e Tempo de Resposta. Uma Requisição é o processo de solicitação de leitura ou de escrita a um dispositivo biomédico, isso por meio da SOA-DB e o Tempo de Resposta é o tempo de processamento de uma requisição ao dispositivo (leitura ou escrita) com a adição do tempo de comunicação na rede, até chegar à aplicação cliente (*Round Trip Time* – Tempo de Ida e Volta).

A Figura 27 ilustra os Tempos de Resposta para um Infusor de insulina hipotético (atuador) implementado na linguagem de programação Java. O tempo médio de resposta para a aplicação cliente em Java, considerando 100 requisições, foi de 178 milissegundos.

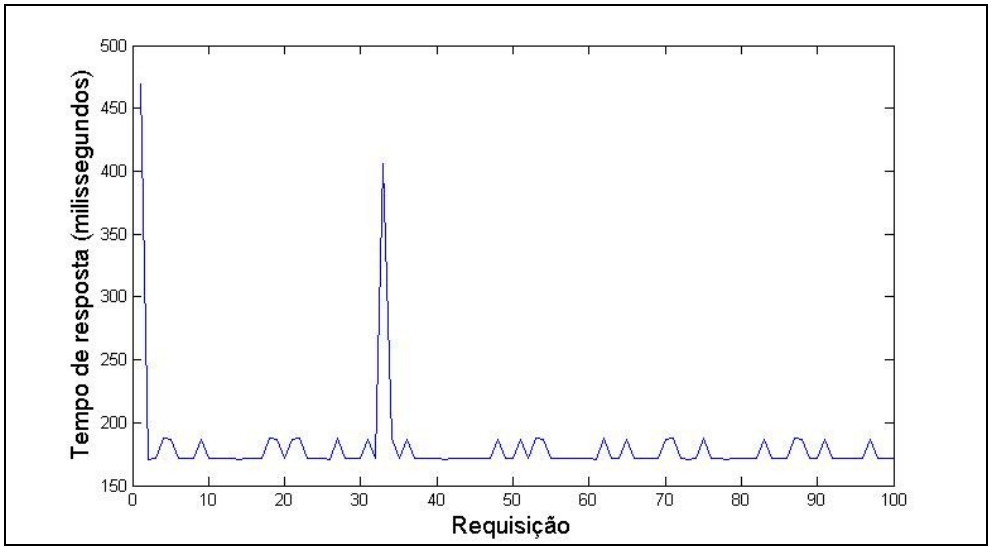

Figura 27: Tempo de resposta para o Infusor de insulina (Aplicação Cliente Java)

A Figura 28 ilustra os Tempos de Resposta para um hipotético Sensor de glicose, implementado na linguagem de programação Java. O tempo médio de resposta para a aplicação cliente em Java, considerando 100 requisições, foi de 1109 milissegundos.

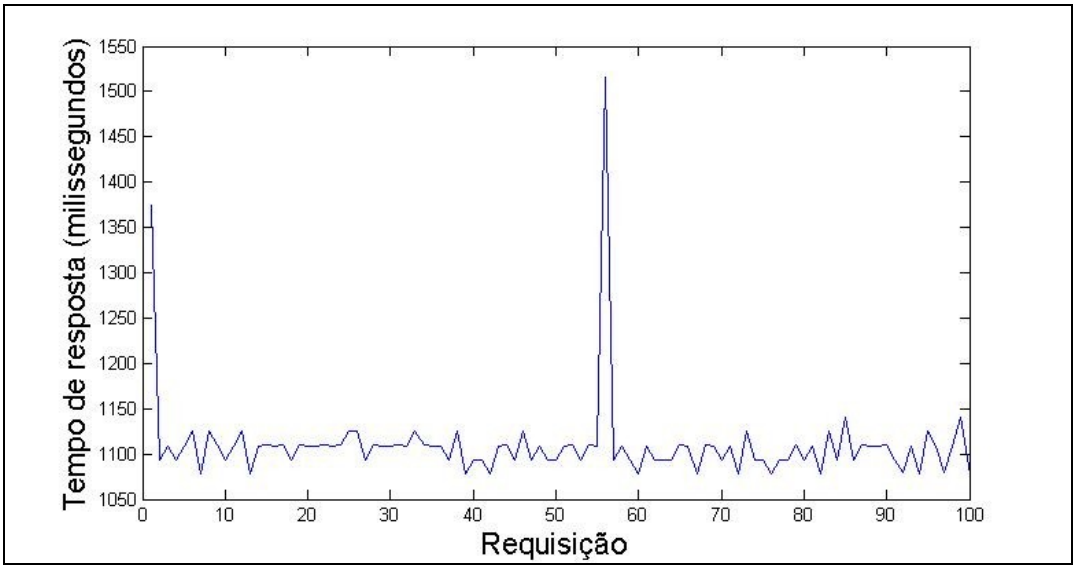

Figura 28: Tempo de resposta para o Sensor de glicose (Aplicação Cliente Java)

A Figura 29 ilustra os resultados para o Infusor de insulina hipotético (atuador) implementado na linguagem de programação C# .NET. O tempo médio de resposta para a aplicação cliente em C#, para 100 requisições, foi de 162 milissegundos.

A Figura 30 ilustra os resultados para o Sensor de Glicose hipotético (atuador) implementado na linguagem de programação C# .NET. O tempo médio de resposta para a aplicação cliente em C#, para 100 requisições, foi de 1079 milissegundos.

A Tabela 4 mostra o Tempo Médio de Resposta, em seguida, o desvio padrão para cada dispositivo e sua respectiva aplicação cliente, com intuito de verificar o quanto os Tempos de Resposta se distanciam do Tempo Médio de Resposta.

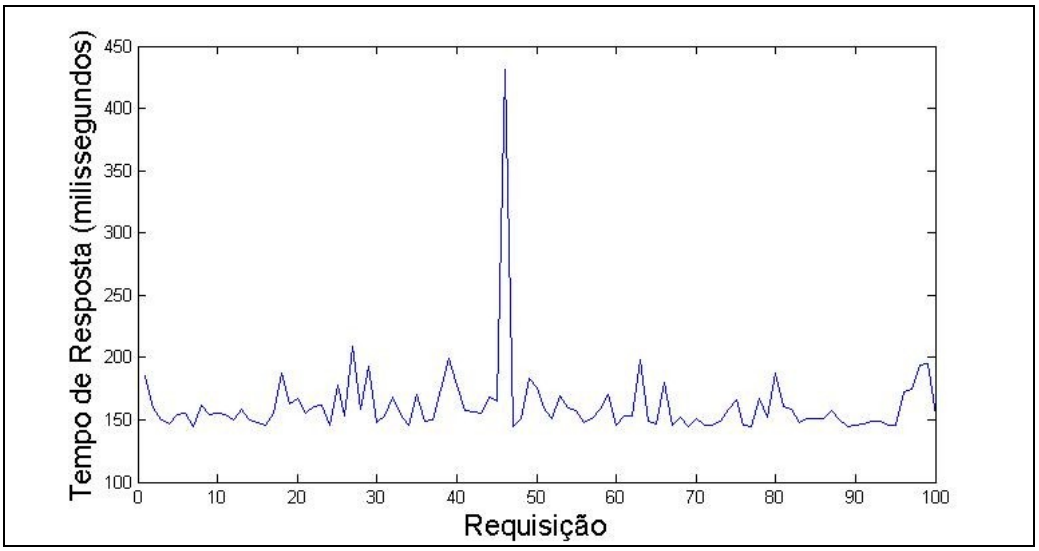

Figura 29: Tempo de resposta para o Infusor de insulina (Aplicação Cliente C# .NET)

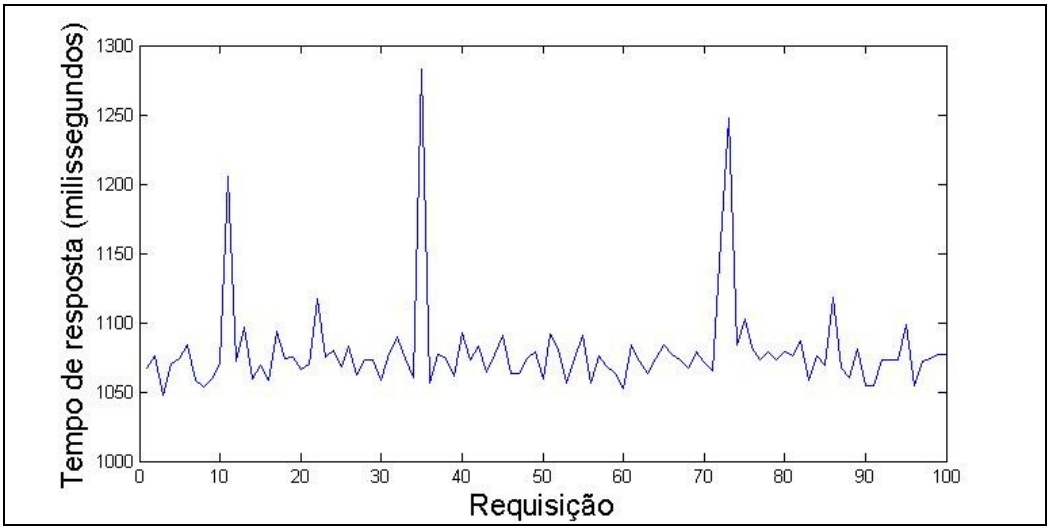

Figura 30: Tempo de resposta para o Sensor de glicose (Aplicação Cliente C# .NET)

Tabela 4: Acesso aos dispositivos com a SOA-DB

Aplicação cliente	**Tempo Médio de Resposta (ms)**	**Desvio padrão (ms)**
Atuador Java	178	37,69
Sensor Java	1109	51,22
Atuador C#	162	30,78

Sensor C#	1079	33,54

A Figura 31 mostra os jitters encontrados, a partir dos Tempos de Resposta obtidos no acesso aos dispositivos requisitados pela SOA-DB, com o intuito de verificar o comportamento dos Tempos de Resposta, ao longo da requisições, conseqüentemente, ao longo do tempo.

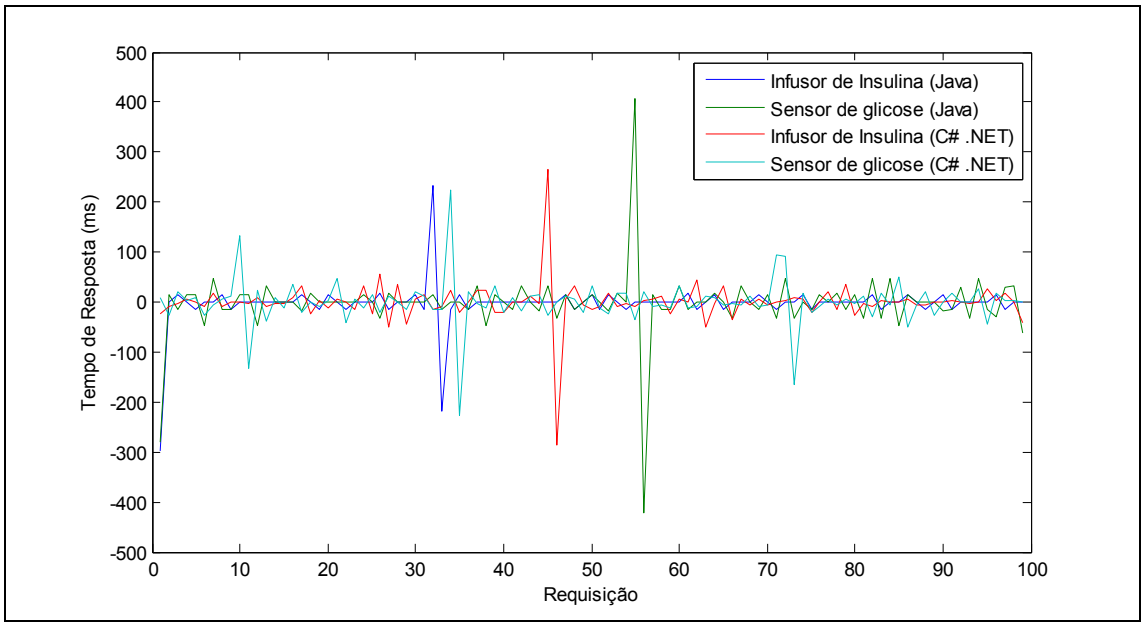

Figura 31: *Jitters* ou variação nos Tempos de Respostas da SOA-DB às aplicações clientes.

3.6.1.2. Discussão dos resultados apresentados

A viabilidade da SOA-DB num ambiente com dados simulados foi constatada, devido à pequena variação nos Tempos de resposta entre os ambientes da Figura 25 (forma convencional de acesso) e das Figuras 27 a 30 (SOA-DB). Essa diferença, para escrita de dados, ficou na ordem de 54 milissegundos, tomando como base o dispositivo mais lento de escrita com a utilização da SOA-DB (o Infusor de insulina hipotético acessando a SOA-DB através de um Cliente Java) e o Tempo de Resposta do infusor de insulina senso acessado localmente (tempo mais rápido). Já para leitura de dados, a diferença entre os Tempos de Resposta, do dispositivo mais lento que acessou

a SOA-DB (o Sensor de glicose Hipotético implementado em Java) e o Sensor de glicose acessado localmente (mais rápido), foi de 79 milissegundos.

Em relação à natureza da linguagem de programação utilizada na implementação das aplicações clientes, foi constatada uma pequena rapidez, das implementadas em C# .NET em relação às implementadas em Java, sendo o Cliente C# .NET 16 milissegundos mais rápido do que o Java, na escrita de dados e 30 milissegundos mais rápido na leitura. Esse fato pode ser explicado pela diferença de plataformas nas quais cada linguagem executa, no caso da linguagem Java, a utilização da Máquina Virtual Java (Que faz uso de uma camada de software a mais que as outras linguagens normalmente compiladas, através da conversão do código programado em *bytecodes* e só posteriormente traduzido em linguagem de máquina, para maiores detalhes, consultar Deitel (2011)) em relação ao .NET *Framework,* plataforma utilizada pela linguagem C# .NET, que executa nativamente no Sistema Operacional Windows. Porém, essa diferença de desempenho tem diminuído consideravelmente ao longo das atualizações da Máquina Virtual Java.

Continuando a discussão acerca dos Tempos de Resposta, foram encontrados alguns picos, que podem ser atribuídos a picos na rede, alternância de contexto nos processadores dos computadores, *bufferização* dos dados, etc., porém com a ocorrência de poucas variações bruscas.

Em relação às operações realizadas com os dispositivos (leitura e escrita), houve diferenças parecidas em ambos os clientes, na medida em que uma operação de escrita é 6,6 vezes mais rápida do que uma leitura, com a utilização da SOA-DB. Já no acesso local (Figura 25), uma escrita é 8,3 vezes mais rápida do que uma leitura. Essa diferença se deve ao fato de que numa escrita, a resposta é retornada para o cliente apenas como um dado *booleano* (se escreveu ou não), já numa operação de leitura, o retorno é o próprio dado requisitado pelo cliente, ao ponto de que uma Requisição de leitura só acaba quando chega ao cliente.

Em relação ao desvio padrão de cada dispositivo, adicionalmente aos seus respectivos clientes, foi constatado um desvio padrão relativamente baixo, visto que o maior desvio padrão (pior caso) foi de, aproximadamente, 21 vezes

menor do que o seu respectivo Tempo de Médio de Resposta (Sensor de insulina hipotético - Java). Paralelamente ao desvio padrão, nos *Jitters*, ou variações dos Tempos de resposta, das requisições dos clientes à SOA-DB (Figura 31), foi observada uma boa **estabilidade**, ocorrendo pequenas variações nos Tempos de Resposta, atribuindo a qualidade de **segurança** para a SOA-DB, já que ambas andam juntas no quesito “transmissão de dados”.

3.6.2. **Ambiente 2 - SOA-DB na arquitetura x86 (Dados emulados)**

- Configurações de Hardware
    - 02 microcomputadores desktop (Microcomputador2 e Microcomputador3)
        - Configuração: Intel Pentium 4 HT, 3.2 MHz, com 1GB de memória RAM
    - 01 microcomputador notebook (Microcomputador1)
        - Configuração: AMD Turion II Mobile, 1.6 MHz, com 2GB de memória RAM
    - 01 cabo de comunicação serial DB 9 null modem.
- Configuração de Software
    - Sistema operacional em uma versão genérica para os 3 (três) microcomputadores;
    - Linguagem de programação Java para o desenvolvimento da SOA-DB;
    - Linguagens Java e C# para implementação dos clientes da SOA-DB;
    - API Java RXTX Serial (Aplicações de acesso aos dispositivos)
    - Apache Axis
    - Sistema de Gerenciamento de Banco de Dados MySQL.

O ambiente da Figura 32 teve um dispositivo real emulado por meio de uma aplicação de escrita de dados (um Eletrocardiógrafo - ECG – emulado)

que teve a função de enviar os dados do paciente (Sinais reais de um paciente, obtidos de um ECG real, consultar em dos Santos et. al. (2010), através de um amplificador customizado (de 2 pólos e passagem de banda de 0.5 a 120Hz) e digitalizados por um microcontrolador PIC com taxa de freqüência de 225Hz. Esses dados foram transformados em um vetor (que continha 1 minuto de batimentos cardíacos) e transmitidos do Microcomputador2 para o Microcomputador3, através da interface RS 232.

No Microcomputador3 foi implantada a SOA-DB.

O Microcomputador 1 se comportou como o Cliente da SOA-DB, realizando as requisições remotas ao ECG emulado. A aplicação Cliente acessou o ECG emulado através de um WSDL, como descrito na arquitetura da SOA-DB (subseção 3.2).

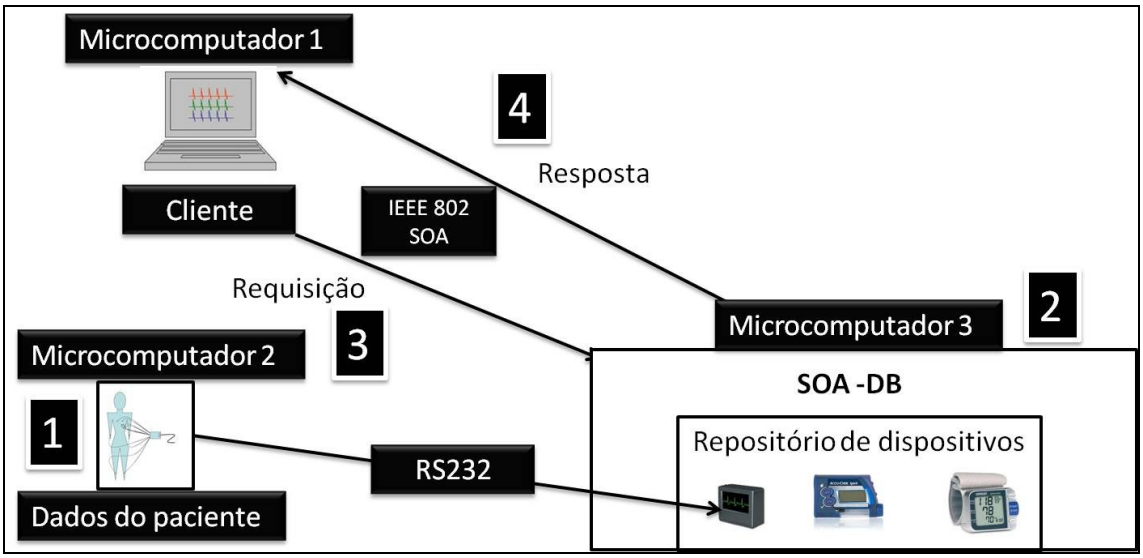

Figura 32: Acesso remoto ao ECG emulado

Fluxo dos testes:

1) Execução da aplicação de transmissão dos dados do paciente
2) Execução da SOA-DB (Aguardando requisições)
3) Requisição da aplicação ao ECG registrado na SOA-DB
4) Resposta da SOA-DB

3.6.2.1. **Resultados experimentais**

A Figura 33 ilustra os dados em que essa subseção de testes se baseou, com os dados coletados em dos Santos et. al. (2010), descritos na subseção anterior.

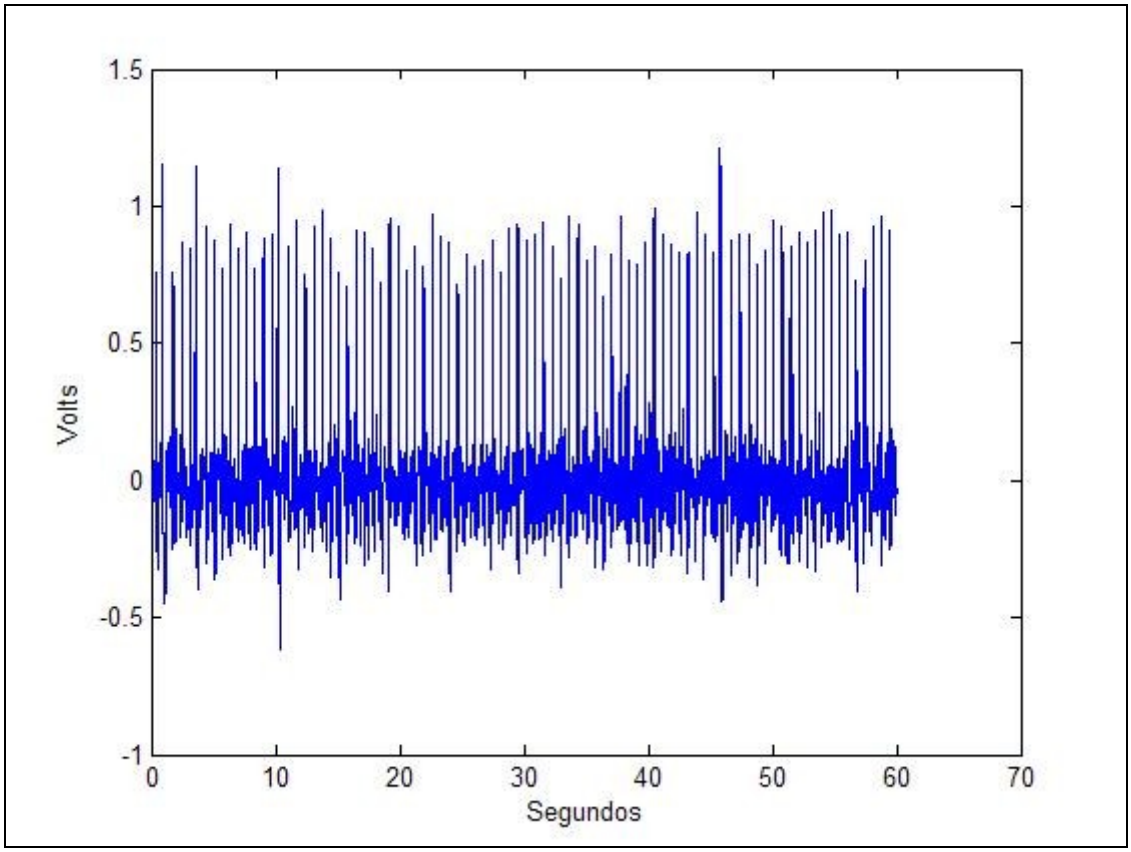

Figura 33: Base de dados reais utilizadas para emulação do ECG.

A Figura 34 apresenta os dados da Figura 30 transmitidos remotamente pela SOA-DB. Foram realizadas 10 requisições pela aplicação cliente, num total de 5 segundos reais de batimentos cardíacos. Ambos os resultados (Figuras 33 e 34) expressam a unidade volts no eixo das ordenadas e segundos no eixo das abscissas.

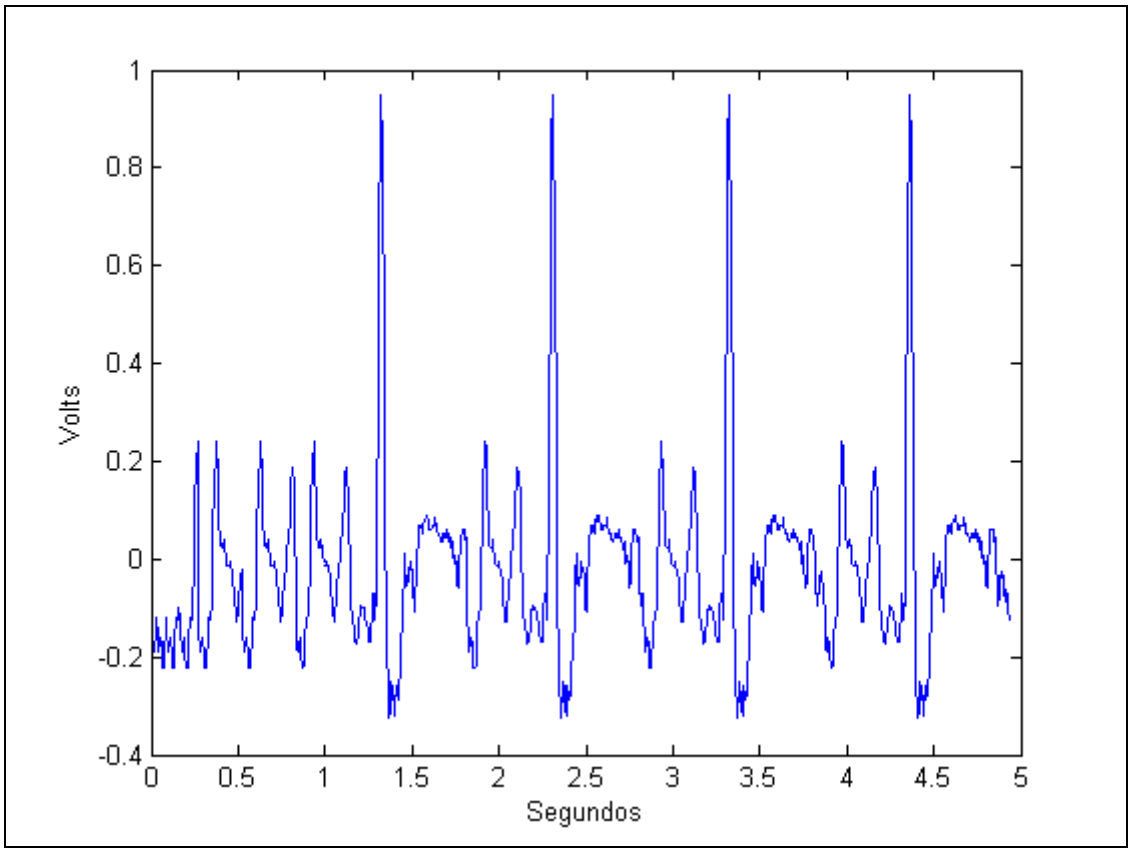

Figura 34: Dados do ECG emulado

3.6.2.2. Discussão dos resultados experimentais

Os testes dessa subseção foram válidos para testar o comportamento da SOA-DB com dados reais de um dispositivo biomédico, na situação, um eletrocardiógrafo. Como pode ser observado na Figura 34, a SOA-DB conseguiu emular de maneira satisfatória os dados de um exame realizado de forma real, um eletrocardiograma, descrito em dos Santos (2010).

3.6.3. **Ambiente 3 - SOA-DB na arquitetura ARM (Dados reais)**

- Configurações de Hardware
    - 01 SoC Beagleboard C4 (BG)
        - Configuração: Processador ARM Cortex A8, 600 MHz, com 256 MB de memória RAM

- 01 microcontrolador de placa-única ARDUINO UNO
  - Configuração: Microcontrolador Atmel AVR de 8bits (ATmega328)
- 01 microcomputador notebook (PC1)
  - Configuração: AMD Turion II Mobile, 1.6 MHz, com 2GB de memória RAM
- 01 cabo de comunicação serial DB 9 null modem
- 01 Microcomputador desktop (PC2)
  - Configuração: Intel Pentium 4 HT, 3.2 MHz, com 1GB de memória RAM
- 01 adaptador USB-Ethernet para a Beagleboard
- 01 Hub USB alimentado
- 01 circuito de sensor de temperatura (ST)
  - Configuração: 01 termístor NTC (*Negative Temperature Coefficient* – Coeficiente de Temperatura negativo) e 01 resistor de 10 KOhms.

- Configuração de Software
  - Sistema operacional Angstrom, arquitetura armv7, para a Beagleboard.
  - Sistema operacional em uma versão genérica Desktop para os PCs 01 e 02
  - Linguagem de programação Java para o desenvolvimento da SOA-DB
  - Linguagens Java para implementação dos clientes da SOA-DB
  - API Java RXTX Serial (Aplicações de acesso aos dispositivos)
  - Apache Axis
  - Sistema de Gerenciamento de Banco de Dados Hypersonic.

A Figura 35 ilustra o ambiente real utilizado neste cenário. Neste caso, o sensor de temperatura é um circuito simples, composto de um termístor ligado em série com um resistor, conectado a uma das entradas analógicas do

Arduino, que realizou a conversão A/D dos dados do sensor. O microcontrolador Arduino, por sua vez, foi conectado à Beagleboard, através de um cabo USB macho-fêmea. Sua principal função foi realizar a conversão A/D dos sinais analógicos do sensor de temperatura. É importante destacar que a porta de saída USB fêmea do Arduino, funciona através de um hardware conversor RS-232 para USB, disponível em sua placa. Toda a comunicação através desta porta é realizada fisicamente pelo padrão serial RS-232.

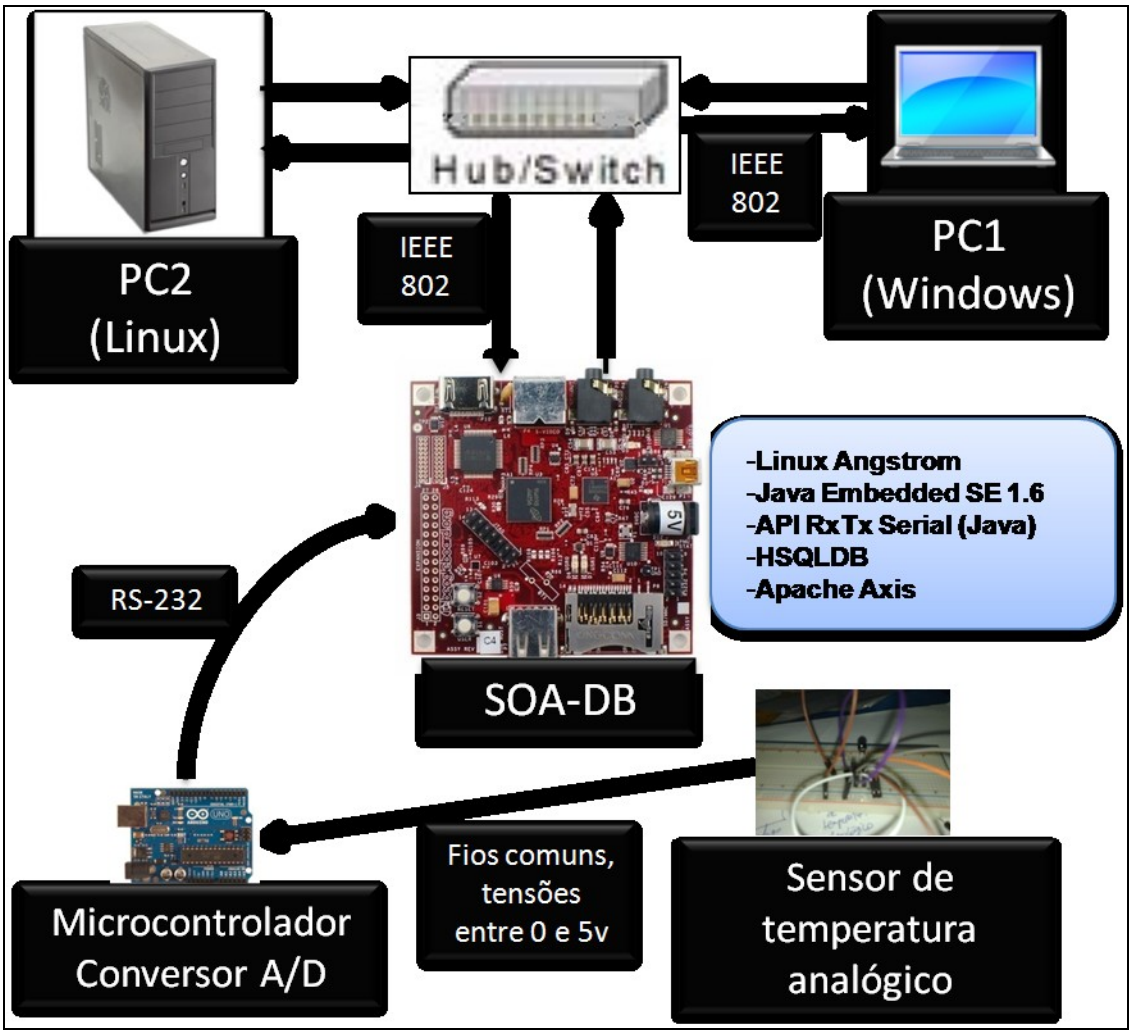

Figura 35: Ambiente SOA-DB na arquitetura ARM

Os dados do sensor de temperatura foram requisitados à SOA-DB através de dois tipos de clientes: Um microcomputador notebook (PC01) e um microcomputador desktop (PC02). Aproveitando-se da portabilidade da linguagem Java, o código fonte de todos os clientes (ver no Apêndice) são

exatamente iguais, variando apenas a máquina virtual Java em cada plataforma, sendo o SO Windows x86 para o PC1 e o SO Linux x86 para o PC2.

Os Clientes requisitaram dados ao sensor de temperatura registrado na SOA-DB, durante 2 (dois) minutos. No primeiro minuto, o sensor de temperatura detectou a temperatura ambiente, já no segundo minuto, foi utilizado um secador de ar quente, com o intuito de observar o comportamento da SOA-DB com variações em seus sensores registrados.

3.6.3.1. **Resultados experimentais**

A Figura 36 ilustra os Tempos de Resposta para o Sensor de temperatura acessado remotamente, pela aplicação Cliente com SO Windows x86, por intermédio da SOA-DB. Os conceitos de Requisição e Tempo de Resposta são os mesmo que foram aplicados na subseção 3.3.1, tanto quanto o Tempo de Resposta é expresso em milissegundos.

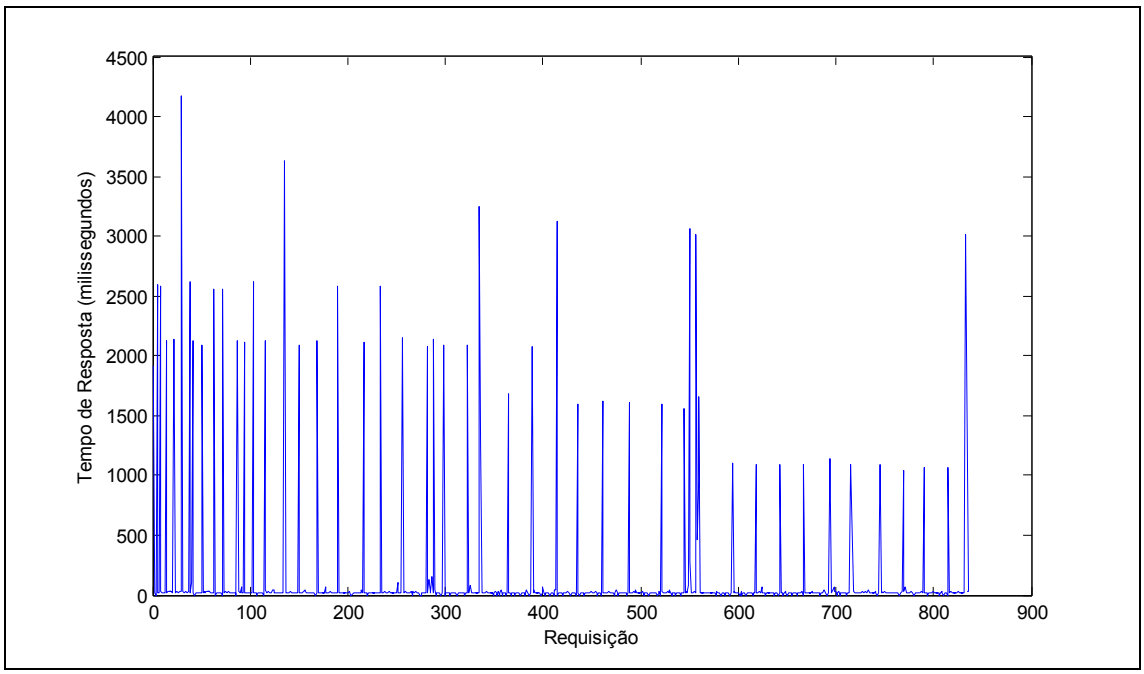

Figura 36: Tempos de Resposta para o Sensor de temperatura acessado remotamente por um Cliente Windows x86.

A Figura 37 ilustra o valor das temperaturas no decorrer do tempo, em milissegundos, para o PC1 (Windows x86). Para obter o instante de cada temperatura, foi utilizado o método Java "*System.currentTimeMillis()*" que

retorna um *long*, com o instante atual, como uma fusão de ano, mês, dia, hora, até chegar a milissegundos. Para fins de visualização, os dados convertidos para segundos.

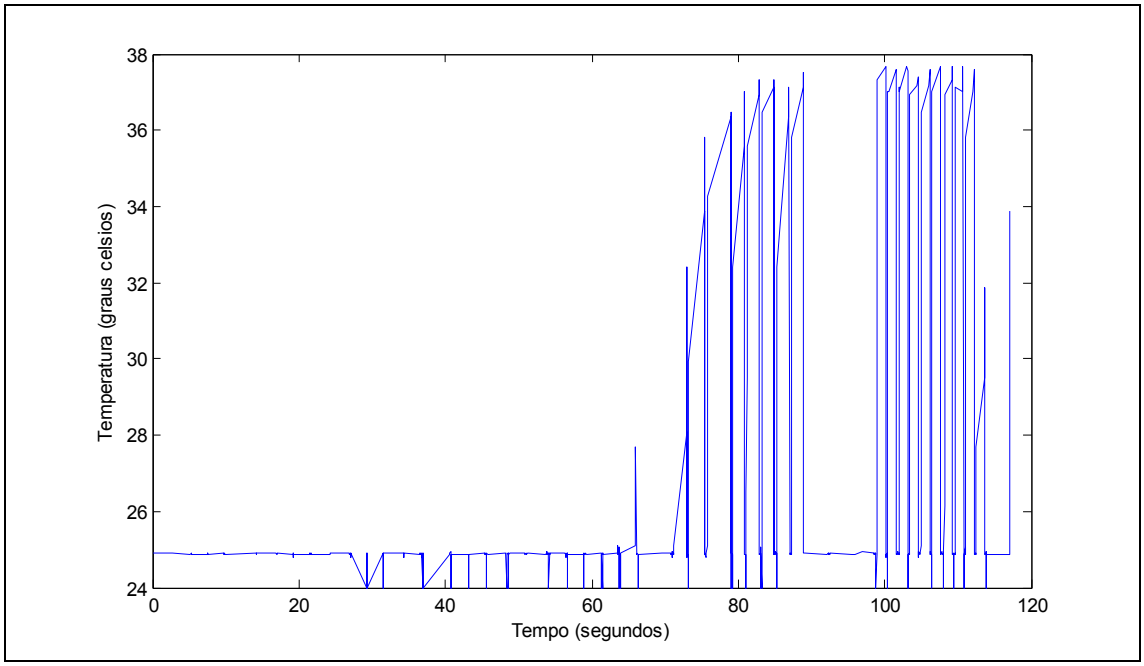

Figura 37: Gráfico de temperatura por segundos do Cliente Windows x86.

A Figura 38 ilustra os Tempos de Resposta para o PC1, com SO Linux x86.

A Figura 39 ilustra o valor das temperaturas no decorrer do tempo, em milissegundos, para o PC2 (Linux x86).

A Tabela 5 mostra a média dos Tempos de Resposta e desvios padrões das requisições ao Sensor de Temperatura, pelos Clientes PC1 e

PC2.

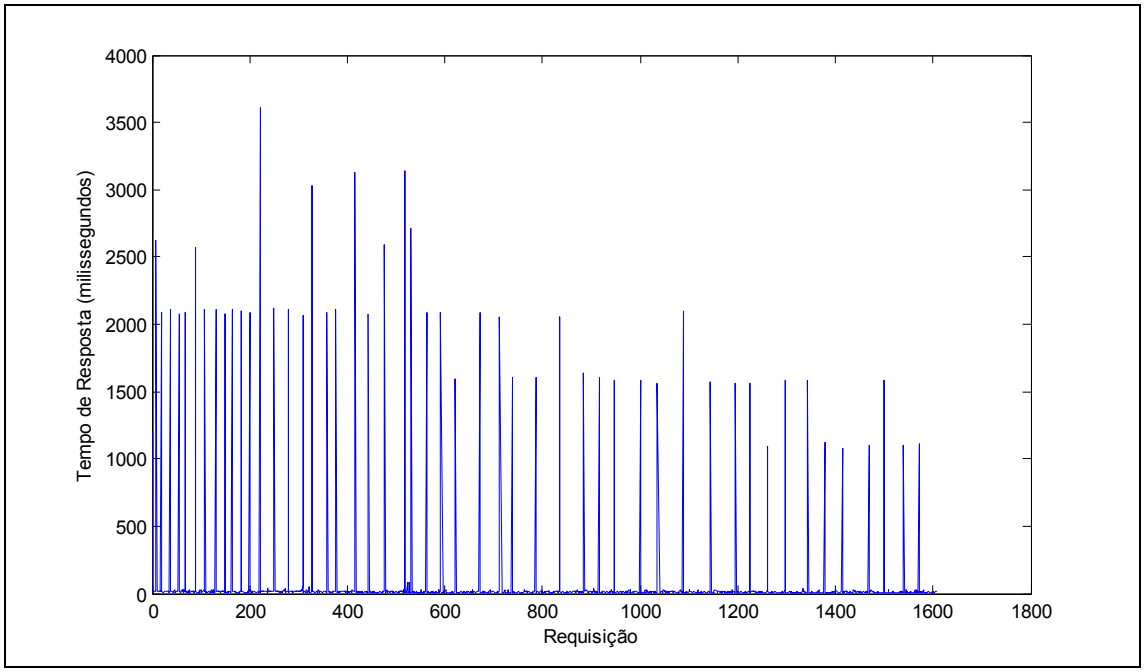

Figura 38: Tempos de Resposta para o Sensor de temperatura acessado remotamente por um Cliente Linux x86.

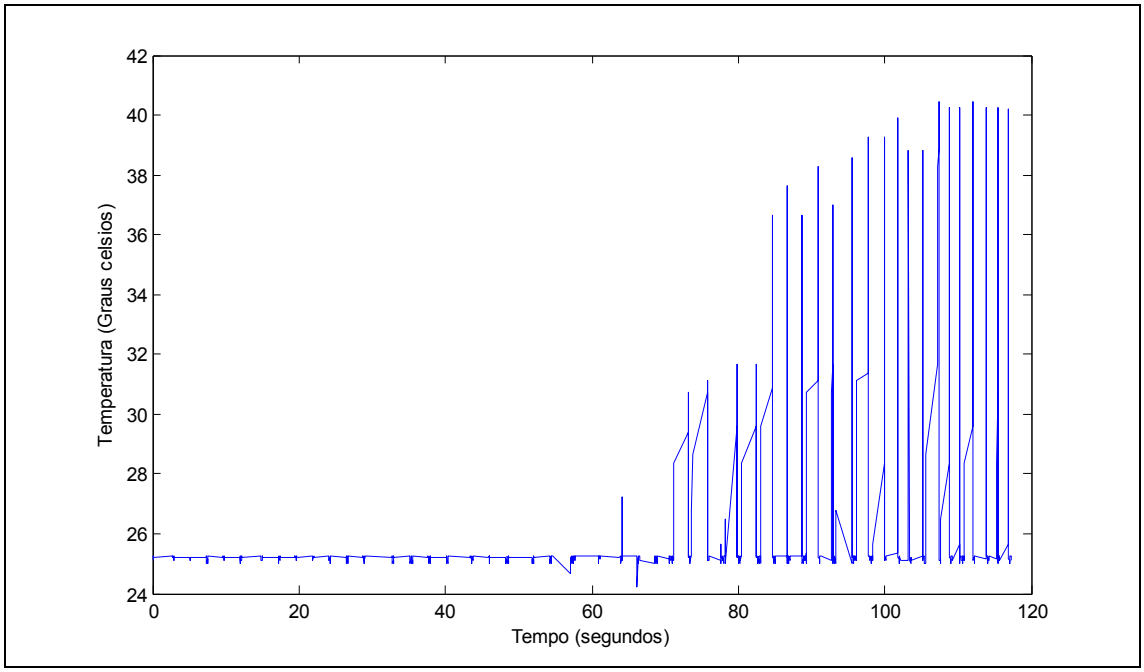

Figura 39: Gráfico de temperatura por segundos do Cliente Linux x86.

Tabela 5: Tempos Médios de Resposta e Desvio padrão para o Sensor de Temperatura

Cliente	Tempo Médio de Resposta (ms)	Desvio padrão (ms)
Cliente Windows x86	142,17	517,55
Cliente Linux x86	73,35	354,15

A Figura 40 ilustra os *jitters* para ambos os clientes (PC1 e PC2), com o intuito de verificar o quanto suas requisições à SOA-DB se aproximaram do desvio padrão dos Tempos de Resposta.

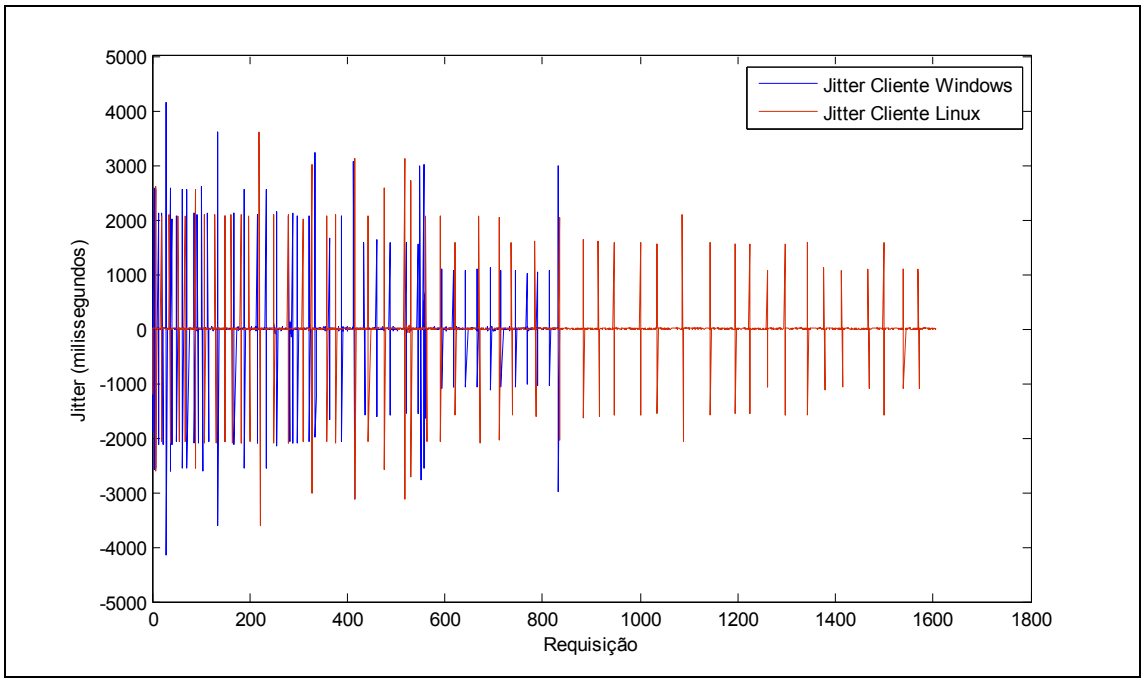

Figura 40: *Jitters* de ambos os Clientes (Windows e Linux).

3.6.3.2. Discussão dos resultados experimentais

Em relação aos Tempos de Resposta obtidos, pôde ser observado que a Aplicação Cliente do PC2 (Linux x86) foi, aproximadamente, duas vezes mais rápida que a Aplicação Cliente do PC1, esse fato pode ser constatado tanto no número de Requisições realizadas, para o mesmo intervalo de tempo, 836 para o PC1 e 1607 para o PC2, quanto nos Tempos Médios de Resposta, 142,17 para o Cliente Windows e 73,35 para o Cliente Linux. Esse fato pode ser explicado pelo fato de o PC2 (Cliente Linux) estar conectado ao *Hub/Switch* por conexão de rede cabeada (padrão IEEE 802.3) e o Cliente Windows estar conectado ao *Hub/Switch* por rede sem fio (padrão IEEE 802.11).

Os Desvios Padrões obtidos no Ambiente 3 (dados reais) foram significativamente maiores do que pois a Resposta dos Dispositivos do Ambiente 1, que acontecia de forma constante e digital, devido ao dado lido ou escrito ter sido o mesmo (um variável texto de 10 bytes). Enquanto que no Ambiente 3, o dispositivo era analógico. O Tempo Médio de Resposta do sensor de temperatura utilizado gira em torno de 30 segundos, ou seja, uma eternidade em relação à velocidade com que os Clientes executaram as requisições, já que isso acontecia em tempo de execução da linguagem de programação.

Os *jitters* encontrados seguem o mesmo raciocínio abordado para os Tempos de Resposta, ou seja, foram bem maiores no Ambiente 3 em relação ao Ambiente 1. O mais importante para os *jitters* do Ambiente 3 é que são bastante parecidos em relação aos Clientes que acessam a SOA-DB, proporcionando estabilidade, da mesma forma que o Ambiente 1. Dessa forma, pode-se inferir que a SOA-DB é segura também para dispositivos reais.

Capítulo 4

Considerações Finais

Desenvolver aplicações para acessar dispositivos biomédicos remotamente não é uma tarefa trivial. Alguns obstáculos aparecem quando está se desenvolvendo, como permissões do SO (Sistema Operacional), requisitos de comunicação entre o SO e a aplicação, ou até mesmo problemas referentes à localização e reconhecimento das bibliotecas de acesso aos dispositivos. Esses problemas podem tirar o desenvolvedor do foco principal: Analisar e codificar o acesso aos dispositivos.

Quesitos como interoperabilidade, integração, encapsulamento das especificidades de comunicação, estabilidade e segurança fornecidos pela SOA-DB, contribuem para a aproximação do desenvolvedor com esse foco principal.

A integração fornecida pela SOA-DB realiza um feito bastante desejado pelos profissionais que trabalham na área de saúde. Ela consegue integrar os protocolos padrões de comunicação do mercado, como RS-232, USB, com os protocolos de comunicação utilizados no ambiente hospitalar, como DICOM ou HL7. Além de fornecer portabilidade, visto que a SOA-DB pode ser implantada em ambientes diversificados sem sofrer alterações significativas.

A partir dos resultados obtidos, a viabilidade prática da SOA-DB foi constatada, após a viabilidade teórica discutida na subseção 3.4, possibilitando um possível investimento na idéia deste trabalho, visto que o conceito da abstração de protocolos de comunicação num ambiente hospitalar foi alcançado.

A diminuição da complexidade no acesso aos dispositivos biomédicos provida pela SOA-DB pode contribuir para o aumento da qualidade nas aplicações de monitoramento de pacientes e diminuição na manutenção de sistemas de monitoramento de dados biomédicos, visto que o desenvolvedor se preocupará com poucos detalhes de acesso aos dispositivos biomédicos,

voltando quase que totalmente a sua atenção para a aplicação, evitando com que os desenvolvedores trabalhem com diversas APIs e protocolos voltados para esses dispositivos.

É importante citar o fato do direito de propriedade intelectual, pois a interoperabilidade fornecida pela SOA-DB não viola de maneira alguma esse direito, visto que determinada funcionalidade do dispositivo biomédico pode ser acessada sem a necessidade de conhecimento de sua implementação.

Este trabalho também serviu para constatar que a adição de uma camada de software a um hardware embarcado não é mais tanto onerosa como há alguns anos. A evolução dos microcontroladores e SoCs ajudaram muito para tornar isso possível, ao ponto do pensamento de performance ao extremo, com a utilização de poucos recursos, tem se tornado obsoleta. Por exemplo, há poucos anos era inviável se pensar em um servidor WEB completo, típico de um hardware potente, ser embarcado.

Portanto, atualmente é possível aliar performance a um baixo custo financeiro, baixo consumo de energia, com a utilização de muitos recursos de software. A Beagleboard é uma prova disso, havendo no mínimo três soluções consagradas de SOs disponíveis para esse SoC, como Ubuntu, Debian ou Android, além de máquinas virtuais Java com quase todos os recursos encontrados nos PCs comuns, bastante superior às versões disponíveis para celulares, além de navegadores *WEB* e capacidade de renderização em 3D.

Um reflexo concreto da evolução tecnológica dos SoCs, pode ser observado nos *smartphones* atuais, que são muito parecidos com os SoCs abordados neste trabalho, os quais realizam diversas funções de um PC típico, com a vantagem da mobilidade e baixo custo. Entretanto, esse potencial tecnológico precisa ser aproveitado em outras áreas, como a saúde, que carece de inovação tecnológica. Este trabalho é uma parte infinitesimal do potencial tecnológico disponível que pode ser aproveitado para inovação tecnológica em saúde, porém é importante para disseminar essa possibilidade.

Em relação ao conceito de Sistemas Embarcados, é preciso acabar com a idéia de separação entre desenvolvedores de sistemas embarcados e desenvolvedores de TI. Ambos podem trabalhar em conjunto e a tendência é que isso convirja cada vez mais.

Uma contribuição muito importante da SOA-DB diz respeito aos fabricantes e fornecedores de equipamentos hospitalares. Ao passo que atualmente, um hospital, por exemplo, ao adquirir um equipamento hospitalar, fica totalmente refém do fornecedor deste, em relação ao treinamento, suporte e reparo. Dessa forma, a SOA-DB pode diminuir essa dependência, visto que, ao invés de se ter um dispositivo com solução *hardware-software* integrada pelo mesmo fabricante, a SOA-DB possibilita a divisão de tarefas na construção do equipamento biomédico.

Neste sentido, surgem tarefas especializadas e independentes na construção de um equipamento biomédico, como o projetista do hardware, o projetista das interfaces de comunicação, o administrador dos dispositivos ou o desenvolvedor da aplicação cliente. O ponto chave é que, uma vez que um determinado dispositivo implemente a SOA-DB, essas funções especializadas poderão se comunicar de forma interoperável, tornando o hospital muito menos dependente de um determinado fabricante ou fornecedor.

Como trabalho futuro é sugerido o desenvolvimento de uma aplicação *WEB*, embarcada na própria arquitetura, para administrar os dispositivos conectados e suas interfaces de comunicação. Isso seria algo muito próximo às aplicações de administração dos roteadores domésticos, pois atualmente, isso é executado de forma manual na SOA-DB.

Outro ponto forte para trabalhos futuros é o quesito Qualidade de Serviço, já que a SOA-DB trabalha num ambiente crítico da saúde.

Referências Bibliográficas

Apache (2005), Web Services – Axis. The Apache Software Foundation.
*http://ws.apache.org/axis/

Apache (2005-2), The Apache Software Foundation, “Apache Tomcat”.
*http://tomcat.apache.org

Arduino (2011), sítio na Internet da plataforma Arduino.
*http://www.arduino.cc/

ARM (2011), sítio da empresa ARM *Holdings.*
* http://www.arm.com/products/processors/index.php

Atmel (2011), AVR Programming Language, Atmel Corporation.
* http://www.atmel.com/products/AVR/

Beagleboard (2011), sítio na Internet da organização Beagleboard.
* http://www.beagleboard.org

De Capua, C.; Meduri, A.; Morello, R.; (2010), "A Smart ECG Measurement System Based on Web-Service-Oriented Architecture for Telemedicine Applications," Instrumentation and Measurement, IEEE Transactions on, vol.59, no.10, pp.2530-2538, Oct.

Deitel, H. M.; Deitel, P. J.; (2003) Java: Como programar; 4a. edição, Bookman.

Deugd, S., Carroll, R., Kelly, K., Millett, B., Ricker, J. (2006) “SODA: Service Oriented Device Architecture,” IEEE Pervasive Computing, vol. 5, no. 3, p. 94-96.

dos Santos, V. L., Prado, R. N. A., Filevich, O, Brandão, G. B., Neto, A. D. D.; (2010) Wavelet-Based Ecg Unsupervised Clustering For Diagnostic Classification, XXII Congresso Brasileiro de Engenharia Biomédica CBEB-2010.

Eggebraaten, T. J. and Tenner, J. W. and Dubbels, J. C. (2007), A health-care data model based on the HL7 Reference Information Model. IBM Systems Journal, Vol. 46, No. 1, pp. 5-18.

Erl, T. (2005), Service-Oriented Architecture: Concepts, Technology, and Design. [S.l.]: Prentice Hall.

Filho, S. J., Pontes, J., Leithardt, V., Multiprocessor System on a Chip (em português) PUCRS.
*http://www.inf.pucrs.br/~gustavo/disciplinas/tppd1/material/TPPDI - Artigo 6 - Julian Pontes Sergio Filho Valderi Leithardt.pdf. Página visitada em 02 de maio de 2011.

Furber, S. (2000). ARM System-On-Chip Architecture (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gamma, E., Helm R., Johnson R., Vlissides J. (1995), Design Patterns. elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

Heath, S.; (1998). Embedded Systems Design. Reading: Butterworth-Heinemann, 1998. 350p.

Imagination (2011), sítio do fabricante de motores gráficos Imagination.
*http://www.imgtec.com/PowerVR/insider/powervr-insider.asp

Júnior, Vladimir Chvojka (2005). Revista Integração (Multidisciplinar) da USJT, vol.XI, no.42: p.251-257. São Paulo. ISSN 1413-6147.

Lacerda, João M. T.; Bruno, Araújo G., Valentim, Ricardo A. M., Guerreiro, Ana M. G., Brandão, Gláucio B., Ribeiro, Ana G. C. D., Soares, H.B.; Araújo, B.G.; Leite, Cicília R. M., (2010), "Mult-I/O - a middleware multi input and output for access devices: A case study applied the biomedical devices," *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pp.3879-3882.

Noumeir, R.; (2003). "DICOM structured report document type definition," Information Technology in Biomedicine, IEEE Transactions on, vol.7, no.4, pp.318-328, Dec. doi: 10.1109/TITB.2003.821334

Papazoglou, M.P. (2003), "Service-oriented computing: concepts, characteristics and directions," Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, vol., no., pp. 3- 12.

Papazoglou, Mike P., Willem J. Heuvel (2007), Service oriented architectures: approaches, technologies and research issues. The VLDB Journal, Vol. 16, No. 3, pp. 389-41.

Schall, D., Marco Aiello (2005) - Web Services on Embedded Devices - J. WEB INFOR. SYST. VOL. 1, NO. 1, MARCH 2005.

Strähle, M., M. Ehlbeck, V. Prapavat, K. Kück, F. Franz, J.-U. Meyer (2007) - Towards a Service-Oriented Architecture for Interconnecting Medical Devices and Applications - Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability.

Texas (2005), *Texas Instruments*, “OMAP2420 Product Bulletin”.
* http://focus.ti.com/pdfs/wtbu/TI_omap2420.pdf.

Vidgen, R. e Goodwin, S. (2000), "XML: What Is It Good For?", vol. 11 , no. 3, pp. 119-124.

W3C (2000). *Simple Object Access Protocol – SOAP*. *http://www.w3.org/TR/soap/.

W3C (2001). Web Services Description Language – WSDL. *http://www.w3.org/TR/wsdl

W3C (2008), World Wide Web Consortium, Web Services Addressing 1.0 - Core. *http://www.w3.org/TR/ws-addr-core Acesso em: Setembro de 2010.

Wang, Chun-Hung et. al. (2007), Development of Intelligent Home Health-Care Box Connecting Medical Equipments and Its Service Platform, em “Advanced Communication Technology, The 9th International Conference” no, vol.1, no., pp.311-315.

Weicker, Reinhold (1984), "Dhrystone: A Synthetic Systems Programming Benchmark" Communications of the ACM (CACM), Volume 27, Number 10, October, p. 1013-1030.

Wikipedia (2010), Sistema embarcado. *http://pt.wikipedia.org/wiki/Sistemas_embarcados.

Wolff, A.; Michaelis, S.; Schmutzler, J.; Wietfeld, C (2007), "Network-centric Middleware for Service Oriented Architectures across Heterogeneous Embedded Systems," EDOC Conference Workshop, 2007. EDOC '07. Eleventh International IEEE , vol., no., pp.105-108.

Zurawski, R. (2004), Embedded Systems Handbook. CRC Press, Inc., Boca Raton, FL, USA.